平成 20 年度

博士論文

ソフトコンピューティングによる自律システムの構造化

福井大学大学院工学研究科
システム設計工学専攻
知識情報システム講座

大蔵 峰樹

目次

1. 序論 5
   1.1. はじめに 5
   1.2. 研究の目的 6
   1.3. 論文構成 7
2. EC サイトにおけるレコメンドエンジンへの遺伝的アルゴリズムの適用 8
   2.1. はじめに 8
   2.2. レコメンド手法 9
      2.2.1. レコメンドの手順 9
      2.2.2. 主に用いられている手法 10
      2.2.3. ソフトコンピューティングを用いたレコメンド 11
      2.2.4. ニューラルネットワーク 12
      2.2.5. 遺伝的アルゴリズム 13
   2.3. 実験内容 15
      2.3.1. 実験方法 15
      2.3.2. 協調フィルタリング方式 17
      2.3.3. 遺伝的アルゴリズム方式 17
      2.3.4. ニューラルネットワークと遺伝子の構成 18
   2.4. 結果 19
      2.4.1. 協調フィルタリング方式 19
      2.4.2. 遺伝的アルゴリズム方式 20
   2.5. 考察 20
   2.6. まとめ 21
   2.7. この章の参考文献 22
3. ニューラルネットワークを応用した Java 言語の難読化 23
   3.1. はじめに 23

3.2. Java 言語と難読化 24
3.3. メソッド名をニューラルネットワークで符号化する方法の提案 27
3.4. 実行結果 30
3.5. 考察 30
3.6. この章の参考文献 31
4. ハードウエアへの遺伝的アルゴリズムの適用 33
4.1. はじめに 33
4.2. 自律行動型ロボットと進化型ハードウエア 34
4.2.1. 自律行動型ロボット 34
4.2.2. 進化型ハードウエア 35
4.2.3. Microbial Genetic Algorithms 36
4.3. 研究内容 37
4.3.1. 小型移動ロボット khepera 37
4.3.2. FPGA タレット 38
4.3.3. 実験フィールド 41
4.3.4. 自律行動型ロボットへの適用 41
4.3.5. FPGA 内部の構成 43
4.3.6. 遺伝子の構成 43
4.3.7. FPGA ソフトウエアパッケージの作成 44
4.4. 実験結果 47
4.4.1. 基本的な評価関数による進化 47
4.4.2. 淘汰圧を加えた評価関数による進化 49
4.4.3. 進化過程での評価関数切り替え 51
4.4.4. 特定の論理演算器のみを用いた進化 51
4.4.5. Microbial GA による進化 53
4.5. 考察 56
4.5.1. 基本的な評価関数による進化 56
4.5.2. 淘汰圧を加えた評価関数による進化 56
4.5.3. 進化過程での評価関数の切り替え 56

4.5.4. 特定の論理演算器のみを用いた進化 57
4.5.5. Microbial GA による進化 57
4.6. 今後の課題 58
4.7. まとめ 58
4.8. この章の参考文献 59
5. 自律移動型ロボットのための集合ニューラルネットワークの進化 61
5.1. はじめに 61
5.2. 集合ニューラルネットワークの進化 63
5.2.1. 進化と協調学習の実装 63
5.2.2. 集合ニューラルネットワークの手順 65
5.3. 実験内容 65
5.3.1. 実験環境 65
5.3.2. 制御ネットワークと遺伝子構成 66
5.3.3. 評価関数とタスク 68
5.3.4. 遺伝的アルゴリズムのパラメータ 69
5.3.5. 実験手順 69
5.4. 実験結果 70
5.4.1. 静的環境における進化と学習過程 71
5.4.2. 動的環境における進化と学習過程 77
5.5. 実験結果の分析 80
5.5.1. 協調学習の強度の効果 80
5.5.2. 障害物によるコントローラーの応答 82
5.5.3. 非協調コントローラーでの評価値 83
5.5.4. 単一ニューラルネットワークと非協調コントローラとの比較 84
5.6. 考察 90
5.7. まとめ 92
5.8. この章の参考文献 93
6. 結論 97

1. 序論

1.1. はじめに

現在，さまざまな課題や問題においてコンピュータを用いた解決が行われており，研究分野に限らずビジネス分野でもなくてはならない存在となっている．数値計算や業務システム，制御分野など，答えが明確であったりやるべきことが決まっている場合も多いが，現実的な問題では答えが明確でなかったり，答えを見つけるためのプロセスそのものが不明な場合もある．また，答えを導くにあたっていくつかの要素が存在し，ある要素を良くするともう一方の要素が悪くなるなど各要素の調整を行い全体として最適な答えを導かなければならない場合などが多い．このような問題の解決方法としては，過去の経験に基づく知識ベースの中から一番近い解を利用したり，問題とその答えのパターン化を行い適用するなどの方法が行われている．近年では，ニューラルネットワークや遺伝的アルゴリズムといったソフトコンピューティングを用いた解決手法が試みられている．

ニューラルネットワークは人間の脳の機能を模したモデルで，パーセプトロンが Rosenblatt によって 1960 年に提案された．しかし，1969 年に Minsky&Papert によって非線形分離問題が解けないという示唆がされたが，誤差逆伝播法（Error Back Propagation Method）が発見され，ニューラルネットワークそのものの研究はもとよりニューラルネットワークを応用した研究が広がった．現在では研究分野に限らず，パターン認識や制御分野において実用的に利用されている．しかし，実用的に利用されているニューラルネットワークは小規模なものが多く，大規模なニューラルネットワークは学習の収束に膨大な時間が必要で実用的に利用するにはいくつかの問題がある．複雑な問題への対応として，大規模なニューラルネットワークとは別のアプローチで，いくつかの小規模なニューラルネットワークを結合し，1 つ 1 つのニューラルネットを特定の問題に対応できるよう学習させそれらを組み合わせる試みが行われている．

一方，遺伝的アルゴリズムは生物の進化過程をモデルとした手法で，John Henry Holland によって 1975 年に提案された．世代ごとに選択，交差，突然変異といった遺伝的操作を行い，問題としている課題に対して最適な解となる個体を進化的に獲得するものである．GA は探索問題や最適化問題など，多様なパラメータが複雑に連携した問題において最適なパ

ラメータの探索を行うのに有用である．近年では，研究・工学分野のほかにもゲームの分野や工業デザインの分野でも活用されている．基本的な遺伝的アルゴリズムでのいくつかの問題点に対応するため，遺伝的操作を拡張した様々な GA の研究がおこなわれている．

現実的な問題での利用が行われているニューラルネットワークと遺伝的アルゴリズムであるが，パターン認識や一部の制御系，最適解を求める問題などへの応用が主となっており，さらなるあたらしい分野への応用が期待される．また，従来的な手法に対して優位性もある反面いくつかの課題も残されており，解決手法自体や他の手法との組み合わせなどによる改善も必要である．

1.2. 研究の目的

さまざまな課題や問題を解決するシステムの構築に，ソフトコンピューティングの手法を用い，問題解決を行うモデルの構造化を目指した．ソフトコンピューティングの手法としては，主としてニューラルネットワークと遺伝的アルゴリズムを利用し，新しい分野での可能性を示唆するために主にソフトウエア関連の分野とハードウエア関連の分野の 2 つにわけ，それぞれで新たな試みを行い提案することとした．

ソフトウエアの分野として 2 つの試みを行う．1 つ目は，EC サイトにおける商品レコメンドエンジンのモデルとしてニューラルネットワークを利用し，既存手法と同様の商品レコメンドが行えるシステムの構造化を試みた．また，ニューラルネットワークの学習方法として，レコメンドを行う商品が同じものにならず意外性のある提案も行えるように遺伝的アルゴリズムを用いて，ユーザの趣向に応じたレコメンドエンジンの自律的構造化を目指した．既存手法では膨大なデータの解析が必要で，ニューラルネットワークと遺伝的アルゴリズムを組み合わせたソフトコンピューティング手法によって，解析時間の短縮とよりユーザに合ったレコメンドエンジンが期待できる．2 つ目は Java 言語のソースコードの難読化の改善を試みた．Java 言語は，ハードウエア等に依存しないクラスファイルと呼ばれるものにコンパイルし仮想マシン上で実行されるが，このクラスファイルの可読性が高い．そのため，ソースコードやアルゴリズムの盗用，セキュリティ的観点で問題となっており，その対策としてソースコードの難読化が行われている．本研究では，メソッド名を

マッピングする構造をニューラルネットワークによる実現を試みた．ニューラルネットワークによるマッピングのため，逆コンパイルを行っても直接的にメソッド名を類推することは困難で，既存手法より可読性が悪い（解析されづらい）難読化が期待できる．

ハードウエアの分野として自律行動型ロボットをテーマに2つの試みを行った.1つ目は，自律行動型ロボットの制御コントローラとして再構成可能なハードウエアを用いて，衝突回避行動を行う制御コントローラの構造化を試みた．また，遺伝的アルゴリズムを用いてハードウエア構成を進化させることにより，進化型ハードウエアとして衝突回避行動の自律的獲得を行う．近年のロボットの制御コントローラは，ソフトウエアで実行されるものが多いが制御が複雑になるほど高性能なコンピュータを搭載しなければならない．ロボットの一部分の反射的制御などを進化型ハードウエアで行うことにより，入力に対する反応速度向上や分散処理が期待される．2 つ目は，自律行動型ロボットの制御コントローラとして，複数の単一ニューラルネットワークをまとめることにより協調動作を行う，集合ニューラルネットワークを用いて制御コントローラとしての構造化を試みた．単一ニューラルネットワークでは，新しい環境変化に伴い，過去の有用な制御モデルに変化が出たり消えてしまったりする問題点の解決が期待できる．

1.3. 論文構成

第 1 章では，本研究に至った背景と目的を述べる．第 2 章では，近年急激な発展を遂げている EC サイトにおける商品レコメンドエンジンとして,ニューラルネットワークと遺伝的アルゴリズムを適用した研究について述べる．第 3 章では，Java 言語におけるプログラムソースやアルゴリズムの盗用，セキュリティ問題への対応のために行われる，難読化の手法としてニューラルネットワークを使った方法の研究について述べる．第 4 章では，自律行動型ロボットの制御コントローラーとして進化型ハードウエアの適用を行い，制御モデルとなる進化型ハードウエア内の回路を遺伝的アルゴリズムによって獲得する研究について述べる．第 5 章では，自律行動型ロボットの制御コントローラーとして，複数の単一ニューラルネットワークをまとめることにより協調動作を行う，集合ニューラルネットワークを適用した研究について述べる．そして，第 6 章にて本論文のまとめを行う．

2. EC サイトにおけるレコメンドエンジンへの遺伝的アルゴリズムの適用

2.1. はじめに

オンラインショッピングサイト（EC サイト）は近年急激な発展を遂げており，カタログ通販・テレビ通販に匹敵する市場規模を形成しつつある．カタログ通販やテレビ通販にはない商品ボリュームの多さによりほしい商品を簡単に見つけることが可能で, 複数の EC サイトを比較検討することで安価に入手できたりなど，ユーザ（お客様）の利便性が非常に高く今後の成長が期待できる．

一般的に商品販売における売上は図 2-1 のようなべき乗則にしたがうグラフを描いており，その店舗における売上の 8 割を全商品のうちの 2 割が占めている場合が多い．販売数量の少ない残り 8 割の部分のグラフが恐竜のしっぽのような形を描いていることから「ロングテール」と呼ばれている．実店舗の場合，在庫の陳列する店舗や在庫を保管しておく倉庫のコストの関係で人気のある商品（図 2-1 における売上が高い 2 割の部分）を多く在庫しておき，人気のない商品（図 2-1 における売上が少ない残り 8 割の部分）はなるべく在庫として持たないのが一般的である．逆に EC サイトの場合，在庫の保管コストが実店舗に比べて安いので, 残り 8 割の部分の商品も少量多品種で幅広く在庫しておくことにより，少ないニーズに多く応えることで売上に貢献することができる．

しかし，商品の品種を多くそろえることにより，商品量が多くなりユーザが目的の商品を探し出すのが困難になる可能性がある．商品を大・中・小カテゴリーに分類し目的の商品を探しやすくしたり，フリーキーワードによる検索によって見つけやすくしたりなどの対策が取られているが，ユーザが欲しいカテゴリーと登録されている商品がミスマッチしていたり有効なキーワードが入力されず商品が埋もれてしまう可能性が高い．そこで，ユーザの趣向の応じて商品をお勧めする（レコメンド）機能を追加する EC サイトが多くなってきており，EC サイトにおけるレコメンド昨日は必要不可欠な要素となりつつある．amazon.com の「この本を買った人は，こんな本も買っています」とおすすめ商品の表示を行っているのが代表的な例である．

現在行われているレコメンド手法の多くは，ユーザ（お客様）がみた商品ページなどの

閲覧履歴を記録しておき，その履歴にも基づいてユーザの趣向を分析しパターン化を行っている．そして，アクセスしているユーザの行動に一番近いパターンを選び出して，その行動パターンに基づきユーザが求めるであろう商品を推測して表示を行う．アクセス数が多いほど多様性が広がりより幅広いレコメンドが行える一方，蓄積される行動履歴データが膨大な量となり解析に時間とコストがかかる．また，パターン化の方法によっては偏りが生じるなど，現在のレコメンド手法には解決すべき問題点が存在する．そこで本研究ではそれらの問題点を解決するべく，ソフトコンピューティングの手法を用いることでまったく新しいレコメンドシステムの自律的構造化を目指す．詳細については，2.2.3 にて議論する．

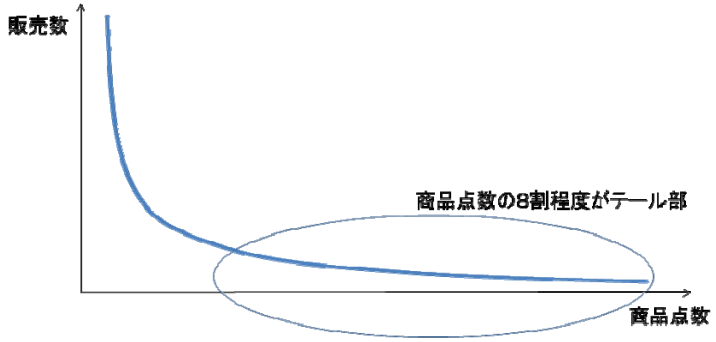


図 2-1 ロングテール

2.2. レコメンド手法

2.2.1. レコメンドの手順

レコメンドはユーザが閲覧したアイテムに関連した，ユーザにとって興味や購買意欲があるアイテムの表示を行う．そのため，多くの場合下記の手順に従って，アイテム同士の関連性を作りレコメンドを行っている．

1. レコメンドを行うための情報を収集する
2. 収集した情報の解析を行いルール・モデル化する
3. 閲覧されたアイテムからルール・モデルに従いマッチングしたアイテムを表示する

1 の情報収集では，サイト内におけるユーザの閲覧履歴（行動履歴）やユーザの属性情報（性

別や年齢，購買履歴など）を主に蓄積する．1 で蓄積した情報を，いろいろな解析手法を用いてルールの作成やモデル化を行う．ルールの作成やモデル化を行ううえで，データ解析の切り口として以下の 2 種類の方法がある．

・アイテムベース

アイテム間同士の関連性に基づいてレコメンドすべきアイテムの抽出を行う．たとえば，ある T シャツを購入している人は合わせてベルトも購入している率が高いものを組み合わせるなど

・ユーザベース

ユーザの趣向に応じてレコメンドすべきアイテムの抽出を行う．あるユーザが特定ブランドのみを購入する傾向がある場合，アイテムの組み合わせなどは考慮せずに特定ブランドのみをレコメンドする．

2.2.2. 主に用いられている手法

現在，商用で利用されているレコメンドエンジンのアルゴリズムは大きく 4 つに分類することができる．

- ルールベース

  ルールベース方式は，特定のアイテムに対してレコメンドを行いたいアイテムの関連付けを行い，閲覧したアイテムに関連付けられたアイテムをレコメンドする．ルールを作る最小単位はアイテムとなるが，取り扱い点数が多いとルール作成に膨大な手間がかかるため，アイテムカテゴリなどによりルールを作成するケースもあるようである．アイテムの販売前にルール作成を済ませておけば，売り手がお勧めしたいアイテムを販売と同時に確実にレコメンドすることができるが，買い手の趣向や行動履歴は反映されず，またルール作成に膨大な手間がかかる．

- コンテンツベース

  コンテンツベース方式は，アイテムの特徴や価格，カテゴリなどから類似値を計算し，

閲覧した商品に類似した商品を検索しレコメンドを行う．この方式もルールベース方式と同様に販売開始前に類似値の計算を済ませておけば，販売開始と同時にレコメンドを行うことができる．しかしながら，レコメンドされるアイテムはどうしても似通ったものとなる場合が多く，たとえば食料品を買いに行って帰りにトイレットペーパーもいかがですか？といったレコメンドを行うことは難しい．

- 協調フィルタリング

  協調フィルタリング方式は，ユーザのアイテム閲覧履歴・購買履歴データを分析・パターン化し，ユーザ同士の趣向の類似値を計算して，類似値の近いユーザの趣向に基づきレコメンドを行う．同じ趣向を持ったユーザの選ぼうとする商品がレコメンドされてくるため欲しい思う確率も高く，また，よく似た商品でも普段選ばないメーカーのものがレコメンドされたり，他のユーザの趣向から新たな商品をレコメンドされる（発見できる）場合もある．

- ベイジアンネットワーク

  ベイジアンネットワーク方式は，ベイズの定理に基づいた統計学的な手法で，さまざまな因果関係で起こった事象の発生確率をもとに推論を行おうとするものである．レコメンドエンジンでは，EC サイト上でのユーザの行動履歴やアイテム間同士の関係性などを要素として，それぞれの因果関係と発生確率を計算し，ユーザが求めている商品のレコメンドを行おうとするものである．しかしながら，事前確立から事後確立を求める手法のため，その際のパラメータなどの設定により結果が大きく左右される問題がある．レコメンドエンジンとしては，まだ十分な結果が出ているとはいえず発展途上の手法といえる．

2.2.3. ソフトコンピューティングを用いたレコメンド

2.2.2 で紹介した既存のレコメンド手法は，膨大なアイテム毎にルール作成を行ったり，ユーザの行動履歴の解析を行い商品同士の関連性やモデル化を行うため，ある一定期間の行動履歴データをまとめて解析する必要がある．多くは 1 日分のアクセスデータを夜間に

解析し，翌日，レコメンドエンジンの基礎データに反映する場合が多いようである．非常にアクセスが多いサイトの場合，解析対象となる行動履歴データが膨大なため，解析時間が足りず翌日に間に合わなかったり，大量のデータを短時間で処理するために並列処理を行うための設備投資が必要となる．また，データの蓄積期間が必要なため，販売が開始された直後の商品に関してはルールベースやコンテンツベースなどでレコメンドを行うしかない．EC サイトで取り扱う商品の属性にもよるが，販売直後にすぐに売り切れてしまうような，非常に商品の動きが速いサイトの場合，レコメンドを行うための十分なデータを蓄積できなかったり，解析後のレコメンドデータでお勧めしようとしてる商品が既に売り切れになってしまっていたりする場合がある．

本研究で提案するソフトコンピューティング手法では，レコメンドを行うためのモデルとしてニューラルネットワークを用いた．ニューラルネットワークの学習方法として誤差逆伝播法などいくつかの手法が存在するが，レコメンドという特性上，複数の解が存在することと，局所解に陥らない必要があるため，ニューラルネットワークの各パラメータを遺伝子として遺伝的アルゴリズムで進化させて学習させる手法を用いた．ユーザ自身の趣向を遺伝的に獲得し商品レコメンドを行い，レコメンドされた商品をユーザが選択することで進化するため，アクセス履歴データから解析してパターン化する必要がなく，リアルタイムで自律的にレコメンドシステムの構造化が可能である．また，強調フィルタリングにおける新たな商品の発見といった機能も，遺伝的アルゴリズムにおける突然変異で同様の機能が期待される．

2.2.4. ニューラルネットワーク

ニューラルネットワークは人間の脳の機能を模したモデルで，神経細胞のモデル化を行い同様の信号処理を行おうとするものである．人間の脳内は多数の神経細胞が互いに結びついている．それらの神経細胞は，ほかの神経細胞からの入力信号の総和が閾値を超えたときに，興奮（発火）しほかの神経細胞に信号を出力する．この仕組みをモデル化したのが，図 2-2 で示すニューロンである．

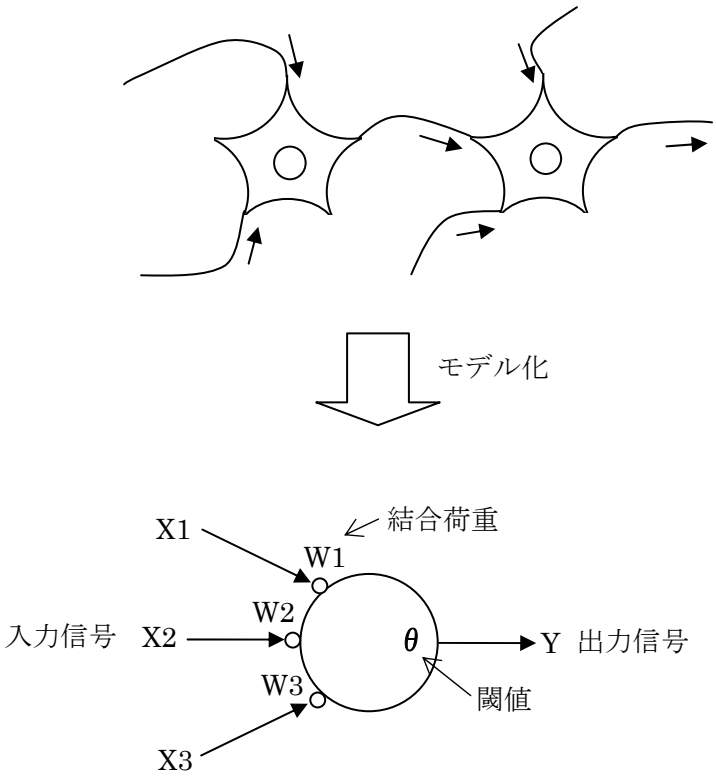


図 2-2 ニューロンモデル

これらのニューロンをお互いに結合させたものがニューラルネットワークで，入力側から出力側への信号の流れが一方向のみの階層型ネットワークと，それに対して信号の流れが双方向に伝わる相互結合型ネットワークの大きく 2 種類に分類される．

2.2.5. 遺伝的アルゴリズム

遺伝的アルゴリズム(Genetic Algorithms:GA)は，生物の進化に見られる過程のいくつかを模倣した，最適化・学習・探索アルゴリズムの一つである．GA では問題の解となるパラメータを遺伝子として表現し，その遺伝子を持つ個体を複数集めた集団を用意する．個々の個体を評価して集団中で淘汰・交差・突然変異などの遺伝操作を行い進化させていく．GA は不確定要素が多いのが欠点となるが，非常に広大な探索空間からより最適な解を早く見つけることができるため，非常に複雑な問題に対する解を求めるといった場合に有効であるといえる．

基本的に GA は淘汰・交差・突然変異の遺伝操作により世代交代をしていく．処理の流れを図 2-3 に示す．まず，最初に初期個体集団をランダムに生成する．このランダムに生成した個体の中に最適な解の一部を持った個体がいることを期待し，集団中で支配力が高まるよう操作していく．まず，遺伝操作を行う前に各個体を評価する必要がある．この評価の際に用いるのが評価関数である．この評価関数によってどのような解を持った個体を得るかが決まるといえる．評価によって得られた評価値順に個体を並び替えて，評価の低い個体は淘汰を行う．残りの集団で交叉を行う．交叉は染色体のある部分で 2 つに分け，別の個体と遺伝子を交換し新しい個体を生成する．また，特定の個体に完全に支配されてしまうことを避けるために遺伝子中のビットを反転するなどの方法で突然変異を行う．遺伝操作を終えた集団を次世代の集団として，評価を行い遺伝操作を繰り返していく．

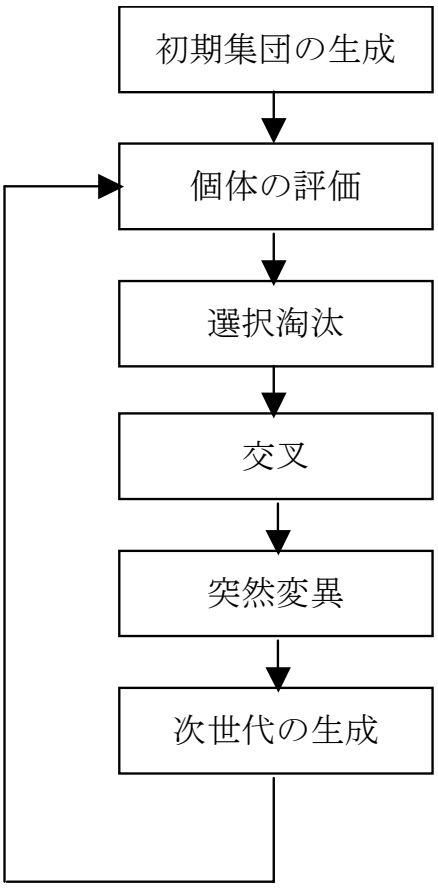


図 2-3 遺伝的アルゴリズム

2.3. 実験内容

2.3.1. 実験方法

レコメンドエンジンの評価としては，基本的には人の好みによる主観的な評価しかなく，評価関数などにより定量的な評価を行うことが難しい．また，実際の EC サイトでは商品ページやショッピングカートのページに，おすすめ商品としてレコメンドの結果を掲載するが，レコメンドを行った商品以外へのリンク用要素も多く，純粋にレコメンド結果の評価を行うことが難しい．そのため，ユーザが閲覧を行った商品に対するレコメンド結果の評価だけを行えるシンプルなページを用意した．図 2-4 にページの構成を示す．

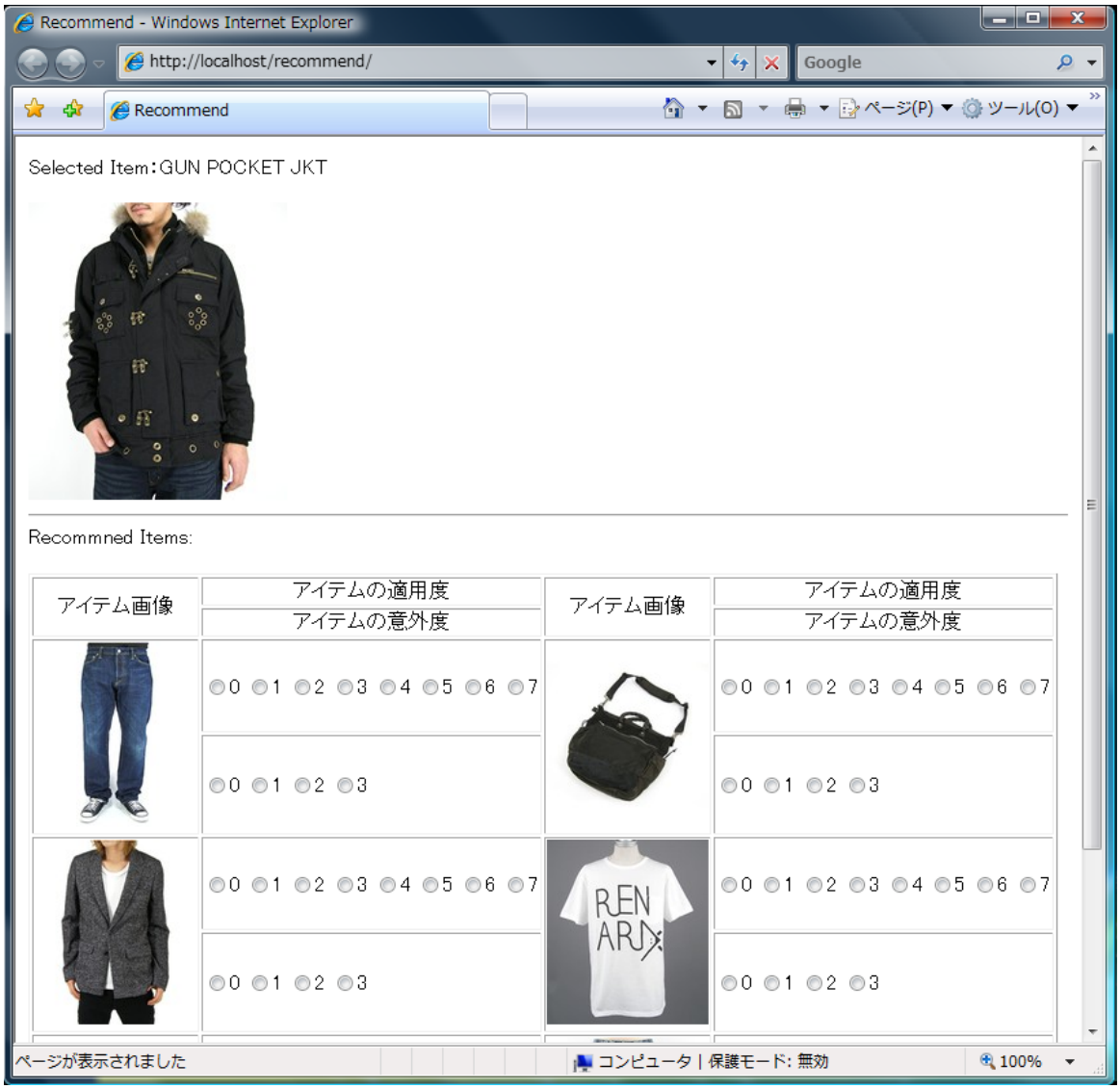


図 2-4 レコメンド画面

ユーザが選択した商品の下部に，レコメンド結果として 10 個の商品の表示を行う．レコメンドを行った商品すべてに対して，欲しい商品であったかどうかと意外な商品のレコメンドであったかどうかの 2 つの観点からそれぞれ評価を行ってもらい，欲しい商品であったかどうかの評価を 0～7，意外性の評価 0～3 の合算を評価値とした．したがって，1 個体の評価値は 0～10 の値をとる．

2.3.2. 協調フィルタリング方式

実際のレコメンドエンジンに用いられている協調フィルタリング方式は，いくつかの手法を組み合わせて行われているが，本研究ではシンプルな相関係数を用いる方法で行った．相関係数は，-1 から 1 までの値をとり，0 に近いほど相関度は弱い．-1 に近いほど負の相関があるといい，1 に近いほど正の相関があるという．相関係数を r とし，商品の閲覧履歴をある一定の数値に置き換えて，ユーザ a のデータ列を a_i，ユーザ b のデータ列を b_i すると次式で与えられる．

$$r = \frac{\sum (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum (a_i - \bar{a})^2 \sum (b_i - \bar{b})^2}} \qquad (2\text{-}1)$$

$\bar{a}$ と $\bar{b}$ は，それぞれデータ列 a_i と b_i の相加平均である．協調フィルタリング方式では，比較すべきデータを事前に作成する必要があるため，比較実験に協力いただいたユーザとは別のユーザに事前に商品を閲覧してもらい，行動履歴データを取得した．

2.3.3. 遺伝的アルゴリズム方式

本研究では，ニューラルネットワークにユーザが閲覧した商品情報を 2 値化した値を入力し，出力も同様に 2 値化した値を取得し該当する商品情報のレコメンドを行う．ニューラルネットワークは，入力層，隠れ層，出力層の 3 階層ニューラルネットワークを使用した．ニューラルネットワークの各ユニット間の結合加重を遺伝子情報として表し，10 個体で 1 世代とした．1 個体に対して 10 回の評価を行い，10 回分の評価値の合計をその個体の評価値として，遺伝ペーレーションを行っている．遺伝子群（世代）はユーザ個別ではなくサイト全体で 1 つとしており，ユーザのアクセスはランダムにあるものとしているため，どのユーザがアクセスしてきているかにかかわらず，サイト全体として商品ページに 10 回のアクセスがあると 1 世代進化するようにした．そのため，10 回分の評価値を合算して 1 個体の評価値としたため，1 個体の評価値は 0～100 の値をとる．

2.3.4. ニューラルネットワークと遺伝子の構成

ニューラルネットワークの各ユニット間同士の結合加重の値をすべて遺伝子情報として持つこととした．1 つの結合加重を 5 ビットで表し，最初の 1 ビットが符号，残り 4 ビットが絶対値とした．本研究で用いたニューラルネットワークは，入力層 10 個，隠れ層 5 個，出力層 10 個となっているので，ひとつの遺伝子は 500 ビットで構成される．

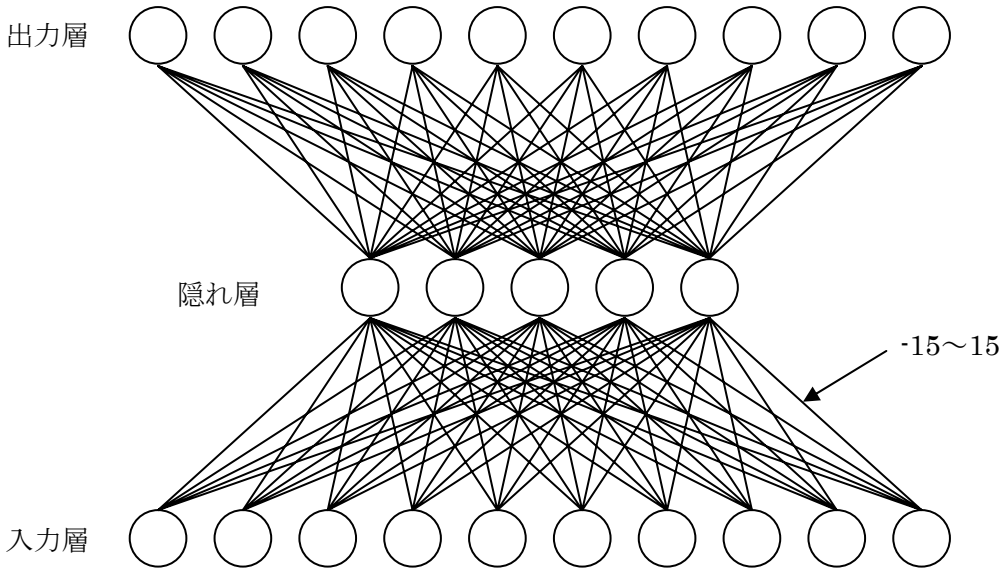


図 2-5 ニューラルネットワークの構成

2.4. 結果

2.4.1. 協調フィルタリング方式

協調フィルタリング方式は事前に収集したデータを元にレコメンドを行っているため，試行中の評価値の向上は見られなかった．本実験では，評価値の理論上の最大値は 100 となっているため，実験結果より平均値が 47.5 となっているため，ほぼ 2 回に 1 回はユーザにあったレコメンドができているか，平均的にまずまずのレコメンドができているといえる．図 2-6 において横軸を generation としているが，便宜上遺伝的アルゴリズム方式との比較のためであり，実際には単純に試行回数を表している．

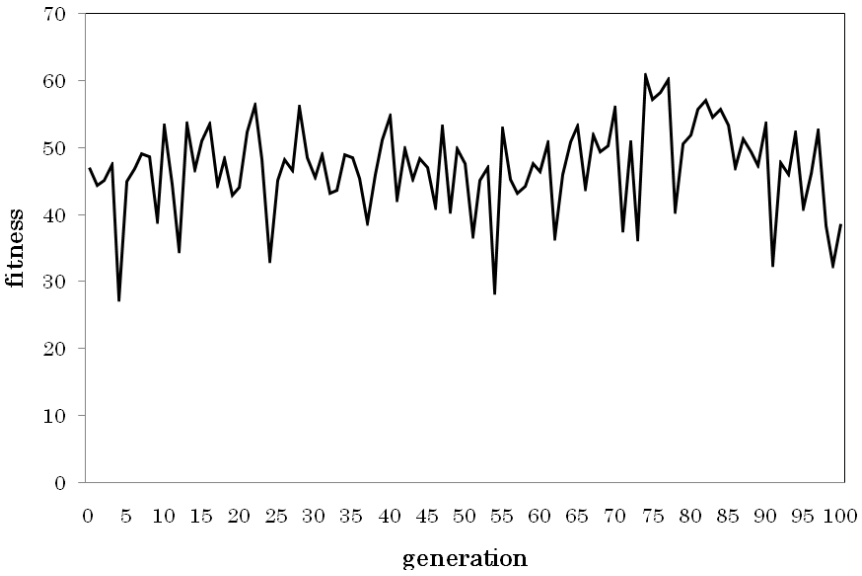


図 2-6 協調フィルタリング方式の試行結果

2.4.2. 遺伝的アルゴリズム方式

遺伝的アルゴリズム方式では，当初は事前情報なしでまったくのランダムでレコメンドが開始されるため，初期段階においては満足なレコメンドは行えていない．徐々にではあるが進化が進むにつれて評価値の向上が見られ，70 世代当たりから評価値が一定となっている．一概に比較はできないが，進化過程の後半において若干劣るものの協調フィルタリング方式と同等のレコメンド能力が獲得できている．

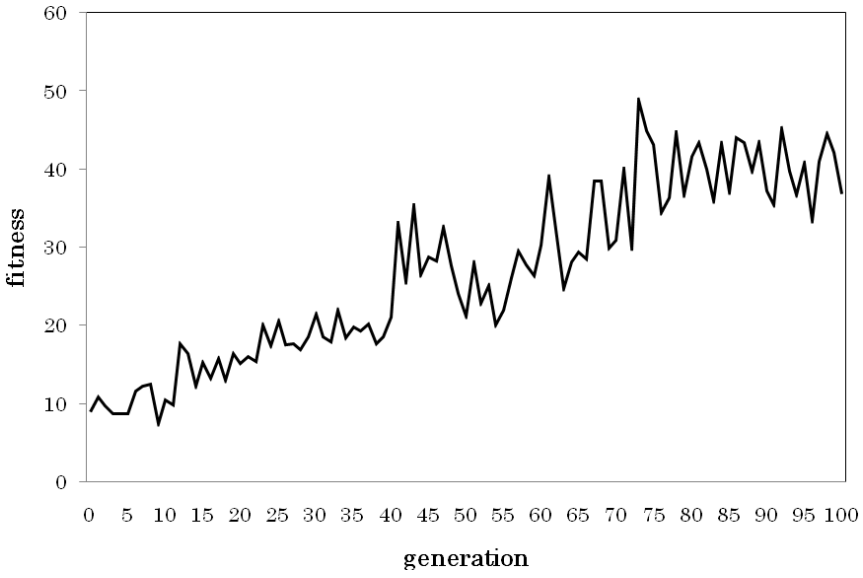


図 2-7 遺伝的アルゴリズム方式の進化過程

2.5. 考察

協調フィルタリング方式は，事前のデータ収集と分析でほぼ成績が決まってしまうため，試行を重ねても評価値の向上はみられない．実際には，リアルタイムに相関係数を算出し，現在アクセスしている他のユーザのデータも分析対象に加えればより高い評価値が得られると考えられる．しかしながら，対象とするデータの範囲をうまく調整しないと，実際の EC サイトへのアクセス時にリアルタイムに数個～十数個の商品をレコメンドするには，十

分に処理能力のあるコンピューターを用意したとしても処理時間的に厳しいと考えられる．

一方，遺伝的アルゴリズム方式は，事前のデータ収集，分析を行っていないため，レコメンド初期においてユーザの求める商品のレコメンドは行えていないが，試行を進めるにつれ徐々にではあるが評価値が向上しており，ユーザの求める商品のレコメンドを行えるよう進化している．進化過程においては協調フィルタリング奉仕域と単純に比較はできないが，70 世代位あたりから同等のレコメンド能力を有したといえる．しかしながら，協調フィルタリング方式に比べて，評価値だけをみると突出して優位なレコメンドを行えているとは言いがたいが，ほぼ同じようなレコメンドを，事前のデータ収集，分析は必要なくリアルタイムでレコメンドモデルを変化して行ける点は優位性があるといえる．

今後の課題として，ニューラルネットワークへの入力データの工夫や，いくつかのニューラルネットワークの構成を試す必要があると考えられる．評価値はユーザの主観による選択を利用したため，ユーザによるばらつきや偏りが出やすいため，より客観的に判断できる評価方法を模索する必要があるかもしれない．この点に関しては，実際に運用されている EC サイトで試行してみることで, 入力データとして利用できるユーザ数とアクセス数を大幅に向上できれば，ユーザによる偏りも解消できるものと思われる．

2.6. まとめ

実際に EC サイトで利用されているレコメンドエンジンは, 単一の算出手法ではなくいくつかの方法を組み合わされて利用されている場合がほとんどであり，遺伝的アルゴリズム方式だけですべてに対応できるレコメンドは難しいと考えられる．レコメンドの的確さだけをみると従来手法とは差ほど代わり映えがしないが，従来手法は基本的に事前にデータの収集や分析，設定が必要になるのに対して，遺伝的アルゴリズム方式の場合は不要であり，優位性のひとつとしてあげられる．また，実験結果よりレコメンド当初は成果を発揮できないが，早い段階で相応のレコメンドを行えるまで進化しており，現在のコンピュータの処理能力があればリアルタイムに進化させていくことも十分に可能で，レコメンドエンジンの 1 手法として有効であると考えられる．従来手法はレコメンドエンジン全体や過去のユーザの行動履歴をモデル化するのに対し，遺伝的アルゴリズム方式の場合，遺伝子

群（世代）を各ユーザ個別に割り当てることによって，似通った商品だけが欲しいというユーザや意外性を求めたいという個別のユーザニーズに答えるといったことも容易に可能である．

本研究によって遺伝的アルゴリズム方式も EC サイトにおけるレコメンドエンジンとして有効な手法として示せたと考えられる．ただ，実際の EC サイトにおいては，いかなる場合でも的確にレコメンドを行えることが要求されており，ニューラルネットワークや遺伝子構成の見直し，第 5 章で述べる集合ニューラルネットワークの利用など，遺伝的アルゴリズム方式自体の改良はもとより，従来手法との組み合わせによってよりユーザの求めに応じたレコメンドエンジンを構築できると考える．

2.7. この章の参考文献

[1] 大杉直樹， 門田暁人， 森崎修司， 松本健一: 協調フィルタリングに基づくソフトウェア機能推薦システム，情報処理学会論文誌, Vol 2004, pp267-278

[2] ICHIKAWA Yusuke, TANAKA Akimichi, KAWAMURA Toru, NAKAMURA Toshirou: The Effect of introduction of Recommender Engine AwarenessNet on online book store bk1, IPSJ SIG Notes, Vol.2005, No.30, pp. 99-104

[3] Keisuke SAHODA, Kenji HATANO, Jun MIYAZAKI, Shunsuke UEMURA: A Web Page Recommendation Method with Collaborative Filtering Using User's Bookmark Hierarchy, DBSJ Letters Vol. 3, No. 1

[4] ONG Mingwei, WATANUKI Keiichi: Development of Specifications Recommendation System for Internet-based Customer Oriented Ordering System, Journal of Japan Society for Design Engineering, Vol.41, No.2, pp. 94-101

[5] 本村 陽一, 岩崎 弘利: ベイジアンネットワーク技術，東京電機大学出版局, 2006

[6] 繁桝 算男，本村 陽一，植野 真臣: ベイジアンネットワーク概説，培風館, 2006

3. ニューラルネットワークを応用したJava言語の難読化

3.1. はじめに

さまざまなシステムで広く利用されているJava言語は，記述されたソースコードをハードウエアやオペレーティングシステムに依存しないクラスファイルにコンパイルし，Java仮想マシンあるいはJavaインタプリタなどのソフトウエアを通じて実行する[1]. したがって，Javaプログラムはプラットフォームの違いを意識しなくても良いが，クラスファイルの中にはソースコードの情報がほとんど残っており，可読性が高くプログラムの解析が容易である．プログラムソースやアルゴリズムの盗用をはじめ，アプリケーションの解析を行うことで重要なシステムや，個人情報や機密情報等のデータベースへの不正侵入など，セキュリティ的にも大きな問題となっている．[2,3]

この問題を解決するため暗号化と難読化の 2 種類の方法でセキュリティ対策がおこなわれている．暗号化は，プログラムファイル自体を暗号化するため配布過程での解読は容易でないが実行前に復号が必要となる．一方，難読化はプログラムの機能を保ったまま，クラスファイルを解析されても解読が難しいものにソースコードを等価変換するものである．難読化は暗号化よりは解読が容易ではあるが，多数のユーザーにプログラムを配布する場合に有用で，難読化ツールも数多く実用化されている[4-10]．難読化の手法としては，変数名やメソッド名などのシンボル名を無意味な文字列に変換する方法[9-10]が古典的であるが，解読が比較的容易である．その他にも数多くの手法が提案されているが[11-16]，現在では，データ難読化と制御フロー難読化が主に用いられている[2]．しかしながらこれらの難読化手法は，解読されにくいもののプログラムの実行時間が長くなったりプログラムが冗長になる欠点があると考えられる．

そこで本研究では，シンボル名を暗号化する方法よりも解読が困難でありながら，実行時間やプログラム長にあまり影響が無い手法として，参照するメソッド名を無意味な 0 と 1 からなる 2 値の数列に符号化する難読化手法を提案する．解読には入力層，隠れ層，出力層の 3 層の階層型ニューラルネットワークを用いる．すなわち，メソッド名が 2 値で入力されるとそれに対応する名前が 2 値コードで出力されるようにニューラルネットワークを予め誤差伝播学習させておく．Java 言語ではメソッドを文字列として直接プログラムに渡

すことが出来ない．そこで，まずニューラルネットワークから出力された 2 値コードを文字列としてリフレクション・アプリケーションプログラミングインターフェース（API）に渡す．リフレクション API は Java クラスからフィールドやメソッドなどの情報を取得する API で，メソッドを文字列で取得・実行することができる．

3.2. Java 言語と難読化

Java プラットフォームは Java アプリケーションプログラミングインタフェース（Java API）と Java 仮想マシン（Java VM）からなっている．Java 言語のソースコードは，ハードウエアやオペレーティングシステム（OS）に依存しないクラスファイルにコンパイルされ，プログラムは Java VM 上で実行される．そのとき，Java API の機能群を通じて Java VM と共にあるコンパイル済みのオブジェクトコードを呼び出す．従って，他のプラットフォームで実行するときにも移植やリコンパイルの必要が無い．クラスファイルには多くの情報が含まれ，その構造は図 3-4 に示すようになっている[17]．クラスの中で定義される変数はフィールドと呼ばれ，また，メソッドは関数とも呼ばれ何らかの処理を担う．Java 言語のクラスファイルは可読性が高く，解析するとデータ構造やアルゴリズムが明らかになり盗用や改ざんが可能となる．そこで，プログラムの解析を困難にする難読化が行われている．難読化はプログラムを解析するのが容易でないように等価変換し，解析に手間がかかるようにして解析行為を無意味なものにすることである．しかしながら，その結果として実行速度が大幅に低下したり実行結果が変化することは許されない．

プログラムの解析を行う方法に逆コンパイラと逆アセンブラがある．逆コンパイラとはオブジェクトコードからソースコードに変換することであるがこれは容易でない．そこで，オブジェクトコードを，それと一対一に対応するバイトコードに変換する逆アセンブラが用いられる．通常，逆アセンブラはプログラムを実行せずに行ういわゆる静的解析で，Java 言語の逆アセンブラを行うと，使用している API やシステムコールに関する情報やデータ内容も得ることが出来る．Java プログラム開発環境で使用されている逆アセンブラとして javap がある．javap で逆アセンブルして得られる情報には，メンバ変数，メソッド，行番号，ローカル変数名がある．各クラスファイルにはそのクラスの各データ・メンバに関す

る全ての命名情報とタイプ情報が含まれる．メソッドの各セクションは，可能な限り，元になったソースコード上の行に対応付けされる．これにより，行番号からスタックのトレースが行える．ある未加工のクラスファイルを javap で逆アセンブルしてバイトコードを取得した結果を図 3-3　逆アセンブルして得たバイトコードに示す．また，上述のようにバイトコードからソースコードに変換した結果を図 3-1 に示す．このように，未加工の Java 言語では比較的簡単にソースコードが明らかになってしまう．

難読化の一般的な方法として，変数名やメソッド名などのシンボル名を意味の無い文字列に変更する方法があり，jmangle [9]や DonQuixote [10]などの難読化ツールでも用いられている．図 3-2 にその一例を示す．このような比較的単純な方法でも，プログラムが大きい場合には解読に手間がかかる．しかし．同じ文字列の変数名やメソッド名は必ず存在するので検索することはできる．また，スーパークラスのメソッドと同じ宣言を持つメソッドをサブクラスで再定義する，いわゆるオーバーライドされたメソッドやインターフェースのメソッドは変換されない[10]．難読化の手法には，その他に複雑な命令を単純な命令の列に置き換えたり，命令を並べ替える方法や[18]，データの符号化と演算の変換による方法[13]，メソッドをクラスファイル間に分散する方法[14]などがある．これらの方法はプログラムを動作させないで行う静的解析によって解読することが可能である．一方，命令コードの実行時置き換えによる方法[12]やデータなどをある演算規則に基づいて変換する方法[13]などでは，プログラムを実行させなければ解読できず，解読にはいわゆる動的解析が必要となる．

```
マジックナンバー(4 バイト)
マイナーバージョン (2 バイト)
メジャーバージョン (2 バイト)
コンスタントプール
エントリーの個数 (2 バイト)
コンスタントプール 1
コンスタントプール 2
        ・
        ・
アクセスフラッグ (2 バイト)
this_class (2 バイト)
super_class (2 バイト)
インターフェースの個数 (2 バイト)
インターフェース 1
インターフェース 2
        ・
        ・
フィールドの個数 (2 バイト)
フィールド 1
フィールド 2
        ・
        ・
メソッドの個数 (2 バイト)
メソッド 1
メソッド 2
        ・
        ・
属性リスト
```

図 3-4　Java 言語の構造

```
public class RANDOM extends java.lang.Object {
   public RANDOM();
   public static void main{java.lang.String[]);
}
Method RANDOM()
   0 aload_0
   1 invokespecial #1 <Method java.lang.Object()>
   4 return
Method void main(java.lang.String[])
   0 iconst_0
   1 istore_2
   2 goto 56
   5 invokestatic #2 <Method double random()>
   8 ldc2_w #3<Double10.0>
   11 dmul
   12 d2i
   13 istore_1
   14 getstatic #5 <Field java.io.PrintStream out>
   17 iload_1
   18 invokevirtual #6 <Method void print(int)>
   21 iload_1
   22 iconst_2
   23 irem
   24 ifne 38
   27 getstatic #5 <Field java.io.PrintStream out>
   30 ldc #7 <String "偶数">
   32 invokevirtual #8 <Method void println(java.lang.String)>
   35 goto 53
   38 iload_1
   39 iconst_2
   40 irem
   41 iconst_1
   42 if_icmpne 53
   45 getstatic #5 <Field java.io.PrintStream out>
   48 ldc #9 <String "奇数">
   50 invokevirtual #8 <Method void println(java.lang.String)>
   53 iinc 21
   56 iload_2
   57 bipush 10
   59 if_icmplt 5
   62 return
```

図 3-3　逆アセンブルして得たバイトコード

```
public class RANDOM {
 public static void main(String[] args) {
  int a;
   for (int i=0; i<10; i++){
    a = (int)(Math.random()*10);
     System.out.print(a);
     if (a%2 == 0) System.out.println("偶数");
     else if (a%2 == 1) System.out.println("奇数");
  }
 }
}
```

図 3-1　ソースコードにリバースした結果

```
public class O__ll__ {
 public static void main(String[] args) {
  int ll__l_;
  for (int Ol__l_=0; Ol__l_<10; Ol__l_++){
     ll__l_ = (int)(Math.randdm()*10);
     System.out.print(ll__l_);
     if (ll__l_%2 == 0) System.out.println("偶数");
     else if (ll__l_%2 == 1) System.out.println("奇数");
  }
 }
}
```

図 3-2　メソッド名を無意味な文字列にした例

3.3. メソッド名をニューラルネットワークで符号化する方法の提案

本研究では，動的解析を必要とするようなメソッド名の符号化を提案する．メソッド名を符号化するとき，その変換対応表を持たせると容易に解読できてしまう．そこで，ニューラルネットワークを実装し，対応表の機能を持たせることでメソッド名の符号化を行う．この場合，変換前のメソッド名に対して変換後のメソッド名を複数にすることが可能である．すなわち，変換後の複数のメソッド名が同じメソッドを意味するという一対多の対応関係を持たせることが可能であり難読化の手法として有用である．

具体的な手順は次のとおりである．

(1) メソッド名の文字を任意の 2 値数値列に符号化する．例えば，A を 10100，B を 10011 のように符号化する．このとき同じ文字に異なる 2 値数列を割り当てても良い．表 1 に例で用いた符号化を示す．入力値は各文字に 1 つの 2 値符号を割り当てるが，文字 A，B，J には 2 つの符号が対応させてある．この例では，出力値はわかりやすいように，文字に対応させる 2 値数列をアスキーコードにしてある．

(2) (1)で決めた対応関係を入出力とするような 3 層の階層型ニューラルネットワークを用意する．その結合荷重は対応関係を教師とする誤差伝播学習によって決定する．入力層，中間層，出力層の素子数がそれぞれ，5，4，5 個のニューラルネットワークに表 1 の対応関係を学習させた後の結合荷重を図 5 に示す．この例では，表 1 に従い，例えば A に割り当てた 2 つの 2 値符号，00110 と 10100，のいずれを入力しても，同じ 2 値符号 00001 を出力する．

(3) プログラムでメソッドを参照するとき，メソッド名を文字ではなく(1)で決定した 2 値符合で記述する．メソッドの実行には，リフレクション API を利用するので，メソッドの修飾子は public としておく．

(4) 符号化されたメソッド名を入力値としてニューラルネットワークを起動し，出力値をメソッド名に変換する．この例では，出力がアスキーコードなので，そのまま文字列に変換する．

(5) (4)で得られたメソッド名を，リフレクション API を介して間接的に実行する．

Method	Code (input)	Character code (output)
A	00110, 10100	00001
B	00111, 01000	00010
C	11001	00011
F	01011	00110
I	10001	01001
J	10011, 00101	01010
K	01111	01011
S	00010	10011
T	11110	10100
X	10110	11000
Y	01101	11001
Z	11011	11010

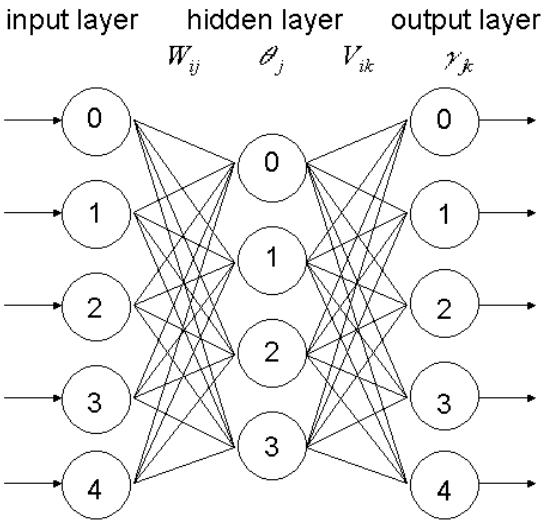
表 3-1　メソッド名と適当に割り当てた符号およびアスキーコード

```
11 newarray int
13 astore_3
14 aload_2
15 invokevirtual #4 <Method void J()>
```

図 3-5　難読化前のバイトコード

```
16 newarray int
18 dup
19 iconst_0
20 iconst_1
21 iastore
22 dup
23 iconst_1
24 iconst_0
25 iastore
26 dup
27 iconst_2
28 iconst_0
29 iastore
30 dup
31 iconst_3
32 iconst_1
33 iastore
34 dup
35 iconst_4
36 iconst_1
37 iastore
38 dup
```

図 3-6　難読化後のバイトコード



W[0][0] = 8.425328935677724
W[0][1] = 2.5813915760511623
W[0][2] = -2.646844393820807
W[0][3] = -12.857707577377234
W[0][4] = 2.6235976695233854
W[1][0] = -11.156466474402192
W[1][1] = -7.762852556765134
W[1][2] = -0.3799894143688571
W[1][3] = 9.554632620486725
W[1][4] = 3.9208238982346493
W[2][0] = 3.2348004553617606
W[2][1] = -5.871583565449208
W[2][2] = 5.7325131403810365
W[2][3] = 2.674135344440894
W[2][4] = 5.4036102582403895
W[3][0] = 4.1076012085441285
W[3][1] = -6.557421398241102
W[3][2] = 8.592691678576848
W[3][3] = 2.7629942298792813
W[3][4] = -7.772697133415665
W[4][0] = -5.752169927157948
W[4][1] = 9.671451479447722
W[4][2] = -13.095051438140144
W[4][3] = -1.2909012087820684
W[4][4] = 1.7354599545341818
theta[0] = -1.3093318526648126
theta[1] = -2.9607411310712624
theta[2] = -8.070898991958845
theta[3] = -2.877431694805768
theta[4] = -4.56682080826006

V[0][0] = 28.687973048806747
V[0][1] = 23.905506053307054
V[0][2] = -25.35310923386969
V[0][3] = 0.690309982555235
V[0][4] = -24.337761586233427
V[1][0] = -10.411864837293894
V[1][1] = 17.926258666094686
V[1][2] = 4.600817939074765
V[1][3] = -27.557038649509686
V[1][4] = 19.998485578110795
V[2][0] = -24.633408509244756
V[2][1] = -7.320515407206533
V[2][2] = -13.77152971482985
V[2][3] = 10.826988371543639
V[2][4] = 16.61314331242105
V[3][0] = -14.19431665369653
V[3][1] = -19.545909210110718
V[3][2] = 13.21318973935469
V[3][3] = -24.454082631214135
V[3][4] = -13.033590289613384
V[4][0] = -18.178912245372086
V[4][1] = -19.845174399390785
V[4][2] = -22.974677612515507
V[4][3] = 5.221114023680544
V[4][4] = -15.225773267341447
gamma[o] = -4.395409166376088
gamma[1] = 1.4645518192996172
gamma[2] = -3.6842443220767915
gamma[3] = 17.299595512280863
gamma[4] = 22.739106343827874

図 3-7　ニューラルネットワークと学習後の係数

3.4. 実行結果

難読化前のプログラムと，提案手法により難読化したプログラムのそれぞれを逆アセンブルした．例として，メソッド J の参照部分をそれぞれ図 6 と図 7 に示す．難読化前のバイトコードでは，ローカル変数からオブジェクトを参照して取り出しスタック積み，参照しているメソッドが J であることがわかる．一方，難読化後では，スタックの一番上の 1 ワード(integer 型の変数)をコピーしてスタックの一番上に積み，integer 型の配列に値を格納する動きが繰り返されている．参照しているメソッド名が J であることはバイトコード上に明示されない．実行時間ついては，当然プログラムによって異なってくるが，我々がテストした範囲ではほとんど影響が無い．長方形と円と三角形の面積を求める簡単なプログラムを 1,000 回実行した延べ時間は，難読化前は 8.060 秒，難読化後では 8.520 秒となった．

3.5. 考察

本研究の提案手法では，プログラムの機能を保持したまま実行効率にさほど影響を与えることなく可読性を低下させることが出来た．この手法で難読化されたものを解読しようとする場合，どのメソッドを呼んでいるかを動的に解析する必要があり容易ではない．実際に利用する際は，メソッドを参照する際の符号化を行ってくれるツールの提供が必要であろう．今後は，他の手法と組合せるなどさらに可読性を下げることなどが考えられるが，現在のところ難読の程度を定量的に評価する基準はなく，プログラムを配布する際にどの程度の難読化が必要かについてもあいまいである．しかしながら，提案手法はニューラルネットワークの応用としては極めて興味深く更に改良の余地があると思われる．

3.6. この章の参考文献

[1] M. Pawlan, Javaプログラミングのエッセンス（藤田怜　訳），(株)ピアソン・エデュケーション，東京， 2002.

[2] 門田暁人，ソフトウエアプロテクションの技術動向（前編）－ソフトウエア単体での耐タンパー化技術－， 情報処理， Vol. 46， No. 4， 2005， pp. 431-437

[3] Y. Sakabe，M. Soshi，A. Miyaji. Java Obfuscation -- Approaches to Construct Tamper-Resistant Object-Oriented Programs， IPSJ Trans， vol.46， No.8， 2005， pp. 2107-2117

[4] DashO， Pre-Emptive Solutions Inc.

[5] ProGuard， by Eric Lafortune， http://proguard.sourceforge.net/

[6] Excelsior JET JVM， XLsoft Co.

[7] Java Code Protector， ChainKey， Inc.

[8] yGuard， (株)ヒューリンクス

[9] jmangle， by Russell Leighton，
http://www.elegant-software.com/software/jmangle/

[10] DonQuixote， by 玉田春昭， http://cafebabe.com/

[11] 門田暁人，高田義広，鳥居宏次， ループを含むプログラムを難読化する方法の提案，電子情報通信学会論文誌 D-I， Vol. J80-D-I， No.7， 1997， pp.1-11

[12] 神崎雄一郎， 門田暁人，中村匡秀，松本健一， 命令コードの実行時置き換えによるプログラムの解析防止， 電子情報通信学会技術研究報告，ISEC02-98， 2002， pp.13-19

[13] 佐藤広紹，門田暁人，松本健一， データの符号化と演算の変換によるプログラムの難読化手法， 電子情報通信学会技術研究報告，IT2002-49， 2003， pp.13-18

[14] 福島和秀，櫻井幸一， メソッド分散による Java 言語の難読化手法の提案， コンピュータセキュリティシンポジウム 2002，pp.191-196

[15] 豊福達也，田端利宏，櫻井幸一， 乱数を用いたプログラム制御構造の難読化手法の提案， 火の国シンポジウム 2005

[16] T. Iwama， An Obfuscation Technique for Java Bytecode Based on Code Scrambling， 第 4 回 SPA サマーワークショップ， 2005.

[17] J. Meyer， T. Downing,（鷲見豊訳）Java バーチャルマシン, O'Reilly Japan， Inc. 1998.

[18] 村山隆徳，満保雅浩，岡本栄司，植松友彦， ソフトウエアの難読化について， 電子情報通信学会技術研究報告，ISEC95-25， 1995， pp.1-6

4. ハードウエアへの遺伝的アルゴリズムの適用

4.1. はじめに

今日のロボットの制御コントローラとしては，ロボットの多様化・複雑化にも柔軟に対応できるソフトウエアによって実現されているものがほとんどである．しかしながら，ソフトウエアを実行するコンピュータをロボットに搭載する必要があり，処理すべき情報が多いほど，高性能なコンピュータを搭載しなければならない．一方，ハードウエアで構成させる制御コントローラは低電力，堅牢性といった点が優れているが，機能の変更が難しくロボットの制御コントローラとしてはソフトウエアでの実装が一般的となっている．

しかし，内部構造を自由に変更できる FPGA(Field Programmable Gate Arrays)を始めとするプログラマブル論理素子の出現で，柔軟なハードウエア制御コントローラの実現が可能となった．また，遺伝的アルゴリズム(Genetic Algorithms:GA)と組み合わせた，進化型ハードウエア(Evolvable HardWare:EHW)により制御コントローラを進化的に獲得することができる．ソフトウエアの場合，ニューラルネットワークなどの制御構造を持っているのに対して，EHW ではその構造自体も自律的に進化によって獲得する．従来我々が考えてきた構造とはまったく違った，新しい構造を獲得する可能性が期待できる．また，回路を進化的に獲得することによって，従来われわれが考えてきた回路とはまったく違った，複雑な処理を非常に少ない部品点数で行なう回路構成を獲得する可能性もある．本研究では，EHW を実際の移動ロボットへ適用して自律的なロボットの制御コントローラとして障害物回避行動の制御構造の獲得を目指した．

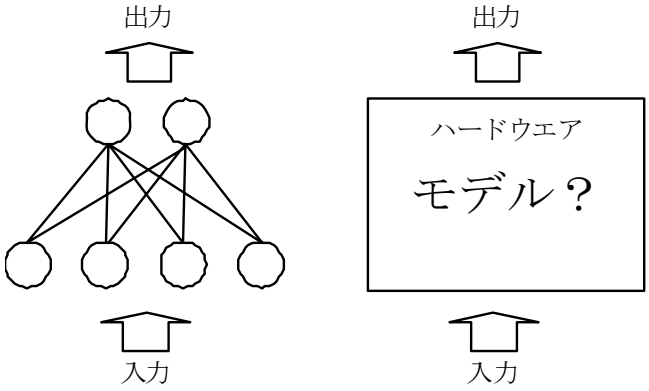


図 4-1 ソフトウエア制御とハードウエア制御

4.2. 自律行動型ロボットと進化型ハードウエア

4.2.1. 自律行動型ロボット

今日，産業などで広く用いられているロボットは，単調な反復作業を正確に高速に処理するロボットである．このようなロボットは，限定された空間・環境で自分の状態を絶えず監視し，人間から与えられた行動モデルにもとづいて行動を行っている．そのため，常に変化する環境下での行動やノイズなどによる予測不可能な事態に柔軟に対応することは不可能である．

このような産業ロボットのように正確性，高速性を目的とせずに，ペットロボットやサービスロボットなど，人間や動物のように常に変化する環境へ柔軟に適応していく能力，ノイズなどの予測不可能な事態にも対応できる能力を持った自律行動型ロボットに高い関心が集まっている．自律行動型ロボットは産業用ロボットのように人間から与えられた行動モデルというものを基本的には持っていない（中には，いくつかの行動モデルを動的に組み合わせることによって，ロボット自らが行動を行なっているかのように動作するものもある）．ロボットは周囲の環境から情報を収集しそれに基づいて自ら行動を決定していく．このような自律行動型ロボットの制御アーキテクチャーとして，行動型 AI と呼ばれる米マサチューセッツ工科大学人工知能研究所のロドニー・ブルックス教授が提唱したサブサンプションアーキテクチャ(SA)が有名である．この SA では単純な行動・反射がいくつも記述

されており，センサー入力に対するモータの行動・反射の創発現象によってロボットの行動が現れる．

また，自律行動型ロボットの行動を遺伝的に獲得していく研究も進められている．ロボットには行動を発生させるための仕組みを与えておいて，その内部状態を進化させて環境に適応させるというものである．ロボットを様々な環境で進化することにより，その環境に応じて適切な行動を取れるようになる．行動を発生させるための仕組みとしてはニューラルネットワークが広く利用されており，その結合荷重を遺伝子として進化させていく方法が取られている．

4.2.2. 進化型ハードウエア

進化型ハードウエア(EvolvableHardware:EHW)は主に 2 つの要素からなる．ひとつはプログラマブル論理素子と呼ばれ，FPGA などに代表される内部の回路構造が書き換え可能（再構成可能）なハードウエア素子である．プログラマブル論理素子は構成情報を外部から書き込むことによりハードウエアの機能・構成を変更することが可能で，ハードウエアの試作品など，製作後にバグなどで機能の変更が必要なプロトタイピングによく用いられる．しかし，FPGA 内部の回路を構成する情報は設計者（人間）によって作成してやる必要がある．

2 つ目は遺伝的アルゴリズム(Genetic Algorithms:GA)と呼ばれる人工知能の探索アルゴリズムである．GA は生物の自然進化に見られるいくつかの過程を模倣した最適化・探索アルゴリズムの 1 種であり，遺伝子と呼ばれる解候補を多数用意し，その遺伝子の適応度を評価関数によって評価を行う．そして，淘汰や交叉，突然変異などの遺伝操作を行い，巨大な探索空間から最適な遺伝子の探索を行う．進化型ハードウエアとは，これら 2 つの要素を組み合わせ，プログラマブル論理素子の構成情報を GA の遺伝子とみなして，その遺伝子を進化させ自動的・自律的に環境や求められる条件に最適なハードウエア構成を発見させるものである．

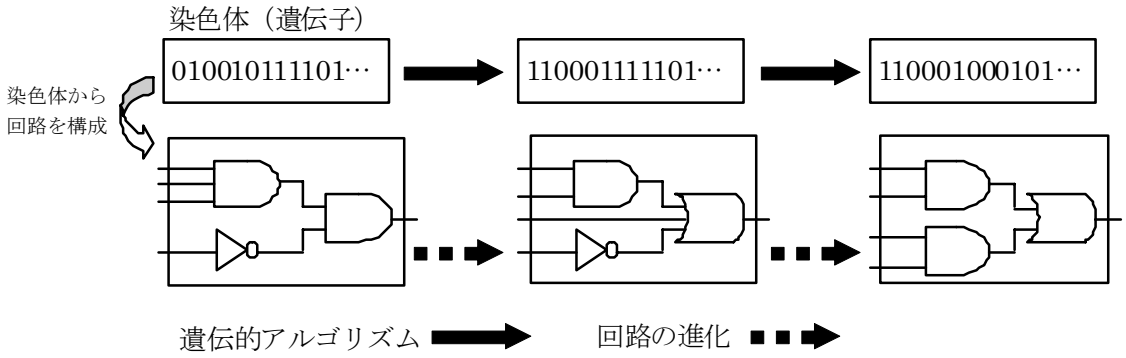


図 4-2　進化型ハードウエア

4.2.3. Microbial Genetic Algorithms

本研究では，2.2.4 で述べた基本的な遺伝的アルゴリズムのほかに，米サセックス大学の Inman Harvey によって提唱された新たな Microbial Genetic Algorithms を用いた実験も行った（Artificial Evolution: A Continuing SAGA (Inman Harvey)， ER2001 LNCS2217 pp.94-109，2001）.Microbial Genetic Algorithms は，微生物の組み換え（伝染）の考え方を導入した GA で，非常にシンプルなアルゴリズムによって実現されている．

Microbial Genetic Algorithms の遺伝操作は，以下の 5 つの操作によって行われる．

1. 集団の中からランダムに 2 つの遺伝子を選択する．
2. 選択した 2 つの遺伝子の評価値を比較し，評価値の高いほうを勝者とし低いほうを敗者とする．
3. 勝者の遺伝子の一部を敗者の遺伝子の一部に書き込む（伝染させる）．
4. 敗者の遺伝子に突然変異を行う．
5. 2 つの遺伝子を集団に戻して新たな世代とする．

このとき，勝者の遺伝子は一切操作されずに敗者の遺伝子のみ操作される．つまり，集団中で遺伝操作が行われる遺伝子は 1 個体のみとなる．Microbial Genetic Algorithms の特徴は，遺伝子は淘汰されずに初期集団の遺伝構造が引き継がれていく点にある．1 世代で変更が行われる遺伝子は 1 個体だけのためゆるやかな進化が行われることになる．

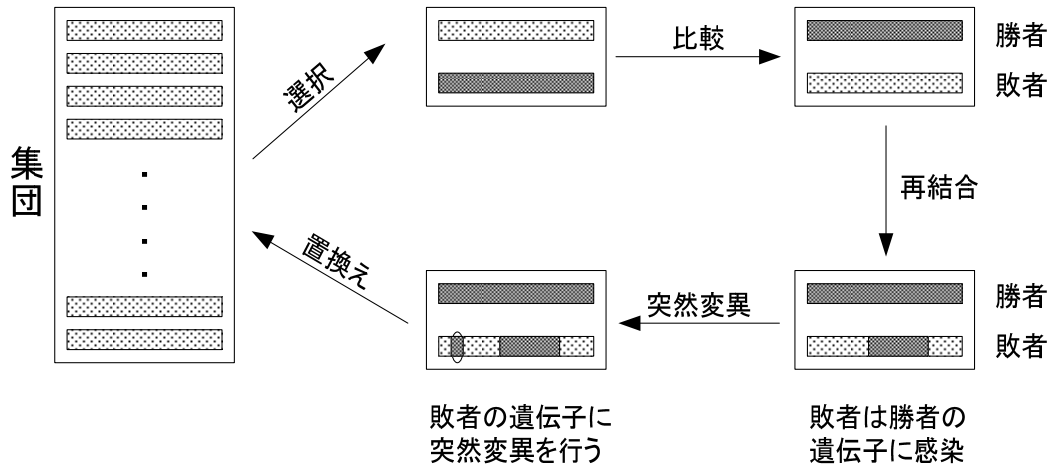


図 4-3　Microbial GA の遺伝操作

4.3. 研究内容

4.3.1. 小型移動ロボット khepera

本研究で用いた小型移動ロボット khepera は，卓上などで手軽に実験が行えるように開発された非常に小さい移動ロボットである．小さいながら，前方 6 個・後方 2 個の赤外線・近接センサー，左右 2 個の独立制御可能なモータ，距離を測定するために左右のモータにエンコーダを備える．外寸は直径 55mm，高さ 30mm，重さ 90g となっている．CPU(MC68331)と RAM，バッテリーを搭載しており，プログラムをダウンロードすることで通信ケーブルをはずしても自律行動が可能となっている．また，ロボットのセンサーやモータを利用するためのプログラムライブラリも用意されており，C 言語でプログラムを行うことができる．

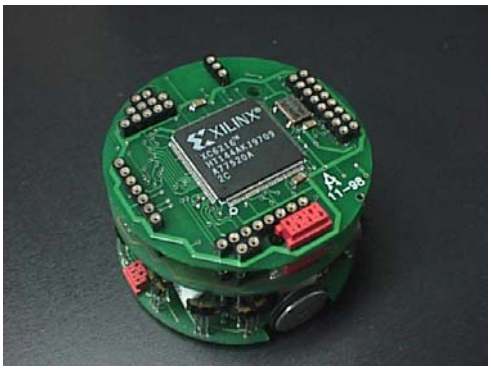


図 4-4　小型移動ロボット khepera

4.3.2. FPGA タレット

khepera にはハンドグリッパーやビジョン(CCD)などの機能を付加できるように「タレット」といわれるオプションボードをロボット上部に装着できるようになっている．本研究では，Applied AI Systems 社（カナダ）が試作を行なった khepera 用の FPGA タレットを利用することができたので，これを進化型ハードウエアとして利用した．

この FPGA タレットには XILINX 社製の XC6216 という FPGA チップが搭載されており，内部が 64×64 の格子状のセル構造（全 4096 セル)になっている．また，東西南北の各辺に I/O ポートがあり （全 256 ポート）それぞれ入力/出力として利用することが出来る．この FPGA タレットで, khepera から実際に利用できる入出力ポートは 14 入力 7 出力が利用可能となっている．

各セルは 4 方向（東西南北）からの入力を受け取ることができ，それを 4 方向に任意に出力することが可能となっている．各セルには 3 入力 1 出力の論理演算器があり，4 方向からの入力を任意に論理演算気の入力とすることができ，出力は 4 方向へ任意に出力ができる（図 6)．論理演算器はその機能を自由に変更することができるようになっており，全部で 18 種類の機能を使い分けることが可能となっている（図 7)．なお，図 4 のロボット上部に見えるチップが FPGA チップである．

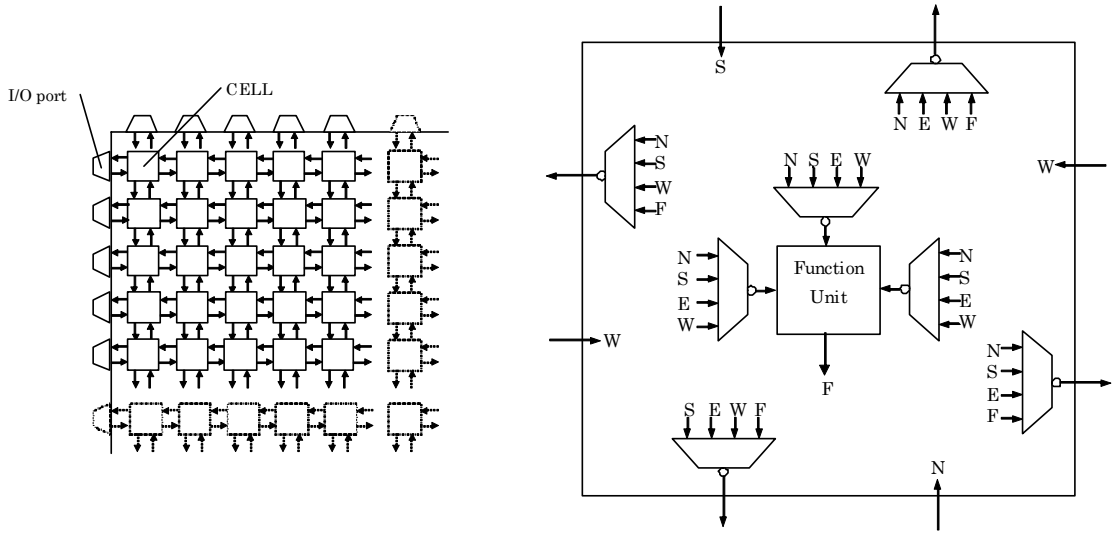


図 4-5　FPGA の内部構造とセルの構造

この FPGA では，セルが格子状に配置されているため回路構成上，かなりの制限を受け柔軟な接続を行なうことは難しい．しかし，論理演算器が非常に豊富な種類を持っておりそのデメリットをカバーすることが可能である．

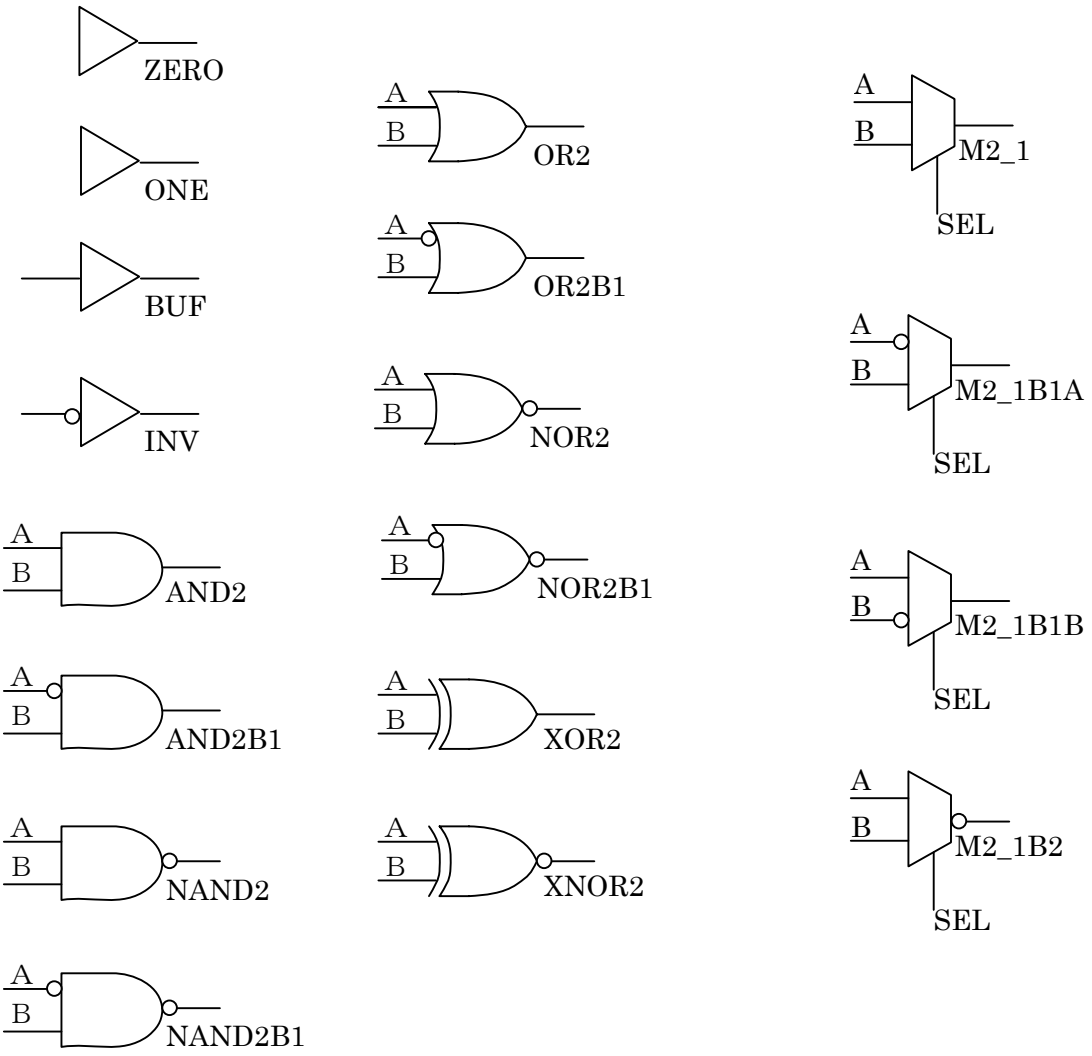


図 4-6　論理演算器の機能

4.3.3. 実験フィールド

実験に用いたフィールドは, アクリルの板の上に図 8 のような回廊状のものを用意した. 照明は特別なものを用いず，フィールド上から普通の蛍光灯の光を照射しただけである.

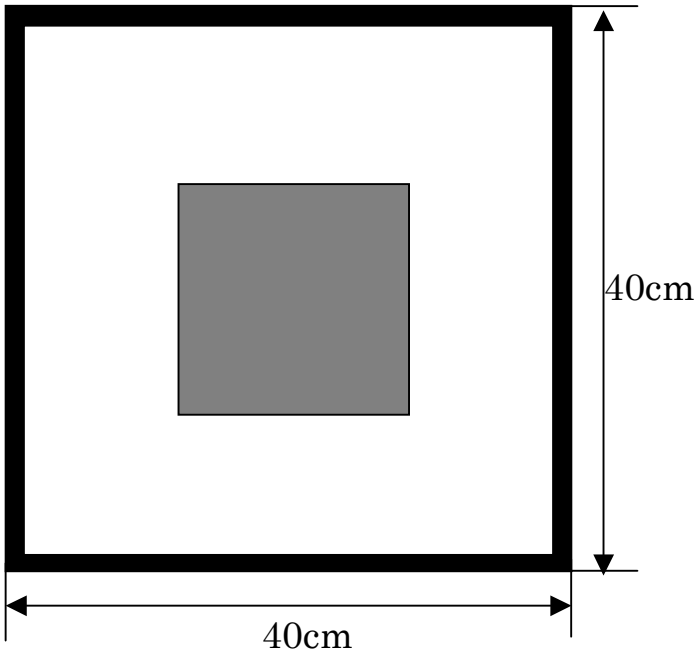


図 4-7 実験フィールド

4.3.4. 自律行動型ロボットへの適用

本研究では, 小型移動ロボット khepera の制御コントローラとして FPGA 上に作成した回路を用い，その回路構成を遺伝子として進化実験を行った．khepera のセンサー入力を FPGA 上の制御回路の入力とし，制御回路からの出力によってモータの制御を行った．FPGA に入力するセンサー情報は，khepera の前方に付いている 4 つのセンサーを使用した．各センサーの場所を図 9 に示す．モータ出力は FPGA 上の制御回路の任意の場所から２個の出力を取り出し，それぞれ左右のモータに割り当てた．進化に用いる評価関数は，従来のニューラルネットワークによる進化などで用いられる壁などにぶつからずにより遠くまで移動するという行動を行うと評価値が高くなる評価関数を基本形として実験を行っ

た．ロボット内の全体的な信号の流れを図 10 に示す.

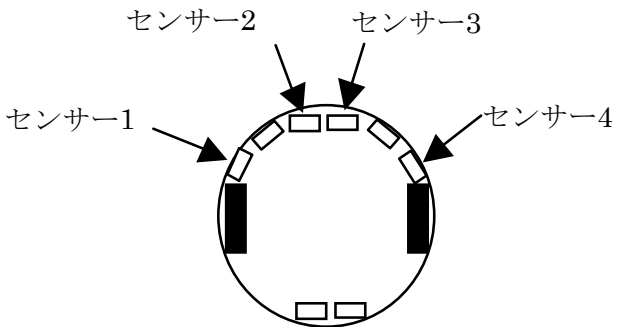


図 4-8 使用したセンサーの場所

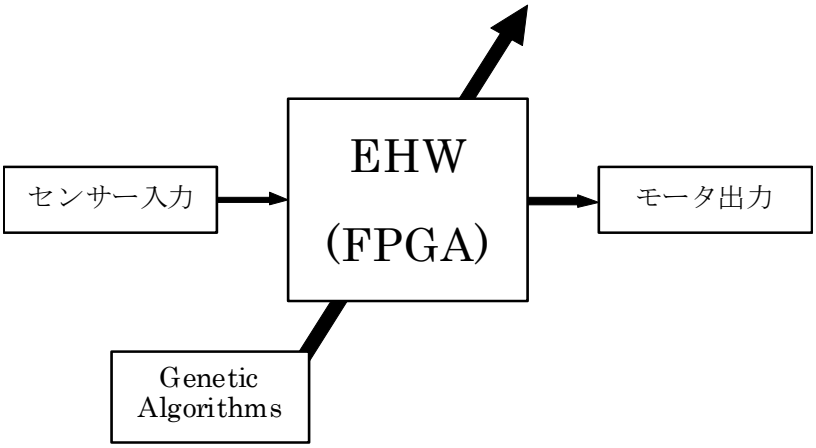


図 4-9　信号の流れ

4.3.5. FPGA 内部の構成

実験では FPGA の内部に 5×5 のセルを用意し，図 11 のように 4 つのセンサー入力と 2 つのモータ出力を用意した．FPGA は論理素子のため，各センサーは任意の閾値で 0，1 の値に変換して FPGA への入力とし，モータ出力は 0 が出力されているときに前進，1 が出力されているときに後進するように設定した．

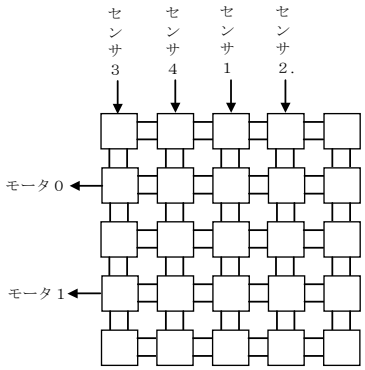


図 4-10　FPGA の内部構成

4.3.6. 遺伝子の構成

5×5 のセルの構成情報を遺伝子とし図 12 のような遺伝子の構成とした．1 セルは，FunctionUnit の機能，FunctionUnit への入力信号，セルの東西南北への出力で 16bit の遺伝子からなる． 1 個体の遺伝子長は 5×5×16 = 400bit となる．

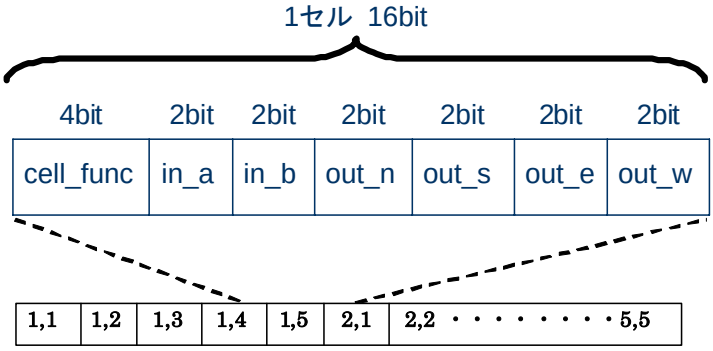
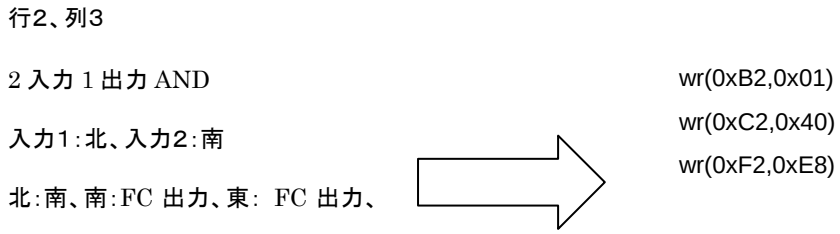


図 4-11 遺伝子構造

4.3.7. FPGA ソフトウエアパッケージの作成

FPGA タレットに付属してくる FPGA の回路構成を書き込むプログラムは，構成を決定する情報を人間の手によって計算した後書き込むものが用意されている．しかしながら，本研究では制御回路を自律的に獲得していくために，遺伝子情報から回路構成情報を生成しなければならない．用意されているプログラムでは，１つのセルの回路構成を書き込む際，セルの機能を３種類に分けそれぞれ対応した組み合わせ表からビット列を抽出し，ビット演算をしてやって FPGA へ書き込むといった方法が取られていた．この機能に対応したビット列の抽出が不規則なものもあり，単に組み合わせ表から抽出するだけでは回路構成データを作成することが出来なかった．



複雑な組み合わせ

そこで，本研究で用いた遺伝子構造をそのまま利用して回路構成を書き込めるようなソフトウエアの開発を行った．具体的には，行と列でセルを指定して，論理演算器の機能，論理演算器への入力，4 方向への出力を指定するだけでそのまま書き込める関数を作成した．なお，論理演算器は 3 入力まで可能であるが，利用できる機能を限定し 2 入力とした．これにより，遺伝子の構造をほぼそのまま書き込むことが可能である．

今回作成した関数は下記ような構成となっている．

□１セルの回路構成を書き込む関数（config_cell）

config_cell(行, 列, FC の機能, FC の入力１, FC の入力２, 北出力, 南出力, 東出力, 西出力)

項目	値
行	1～64
列	1～64
演算器の機能	ZERO　　OR2 ONE　　OR2B1 BUF　　NOR2 INV　　NOR2B1 AND2　　XOR2 AND2B1　　XOR2B1 NAND2 NAND2B1
演算器の入力１ 演算器の入力２ 北出力 南出力 東出力 西出力	North South West East

例：
２行３列目　演算器の種類:AND 演算器への入力:北入力、南入力
北南西東への出力：[北]←南入力,[南]←演算器の出力,[西]←演算器の出力,[東]←南入力

config_cell(2,3,AND2,North,South, South, Function,Function, South)

この関数の内部動作を図 14 に示す．まずこの関数の引数から，「セルの行と列」「演算器の種類」「東西南北への出力」をそれぞれマップに対応させてビット列を抽出する．それら 3 つのビット列を組み合わせて，1 つが 8bit の 3 つのデータを作成する．特殊な組み合わせの場合の例外処理を行ってから，FPGA のモードを書き込み状態に変更してデータの書込みを行う．

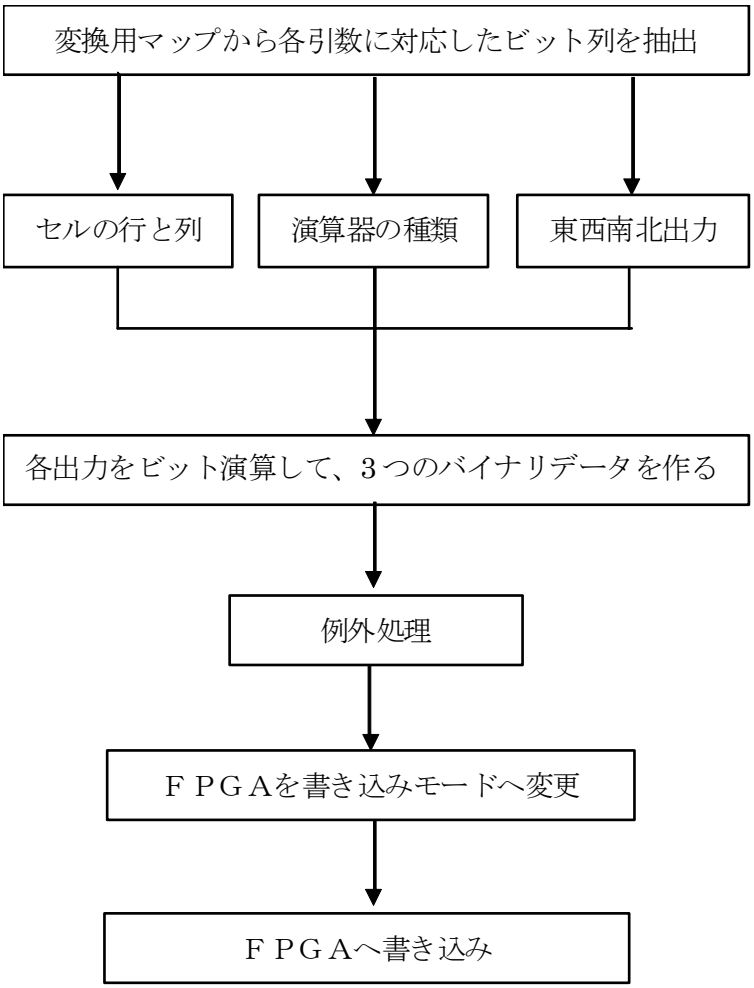


図 4-13　書込み関数の内部処理

4.4. 実験結果

4.4.1. 基本的な評価関数による進化

評価関数として，できるだけ壁に近づかずに長い距離を移動できれば評価値が高くなるものを使った.

【評価関数】

$$F = D \times (1\text{-}S)$$ [F:評価値　D:移動距離　S:センサ入力]

【結果】

初期段階

- 前進のみ，後進のみする
- その場で回転する

30～50 世代

- 前進のみ，後進のみする
- 左右の回転を繰り返す

100 世代～

- 前進のみ，後進のみする

進化の初期段階では回転する個体も見られたが，最終的には前進のみまたは後進のみを行う個体に支配された．これは，前後進だけを行っていれば確率的にある程度の評価値を得られるためだと考えられる．この実験では，センサー入力に反応して動きを止めたり，壁を回避するといった行動は見られなかった.

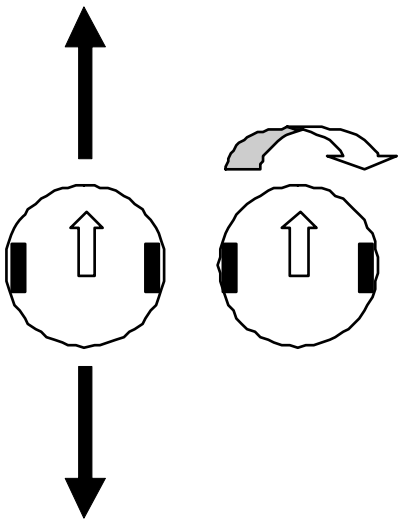


図 4-14　ロボットの動き

【進化過程】

集団中で一番評価値の高いものは徐々に進化しているようだが，集団全体しては 10 世代以降ほとんど進化していないのがわかる．

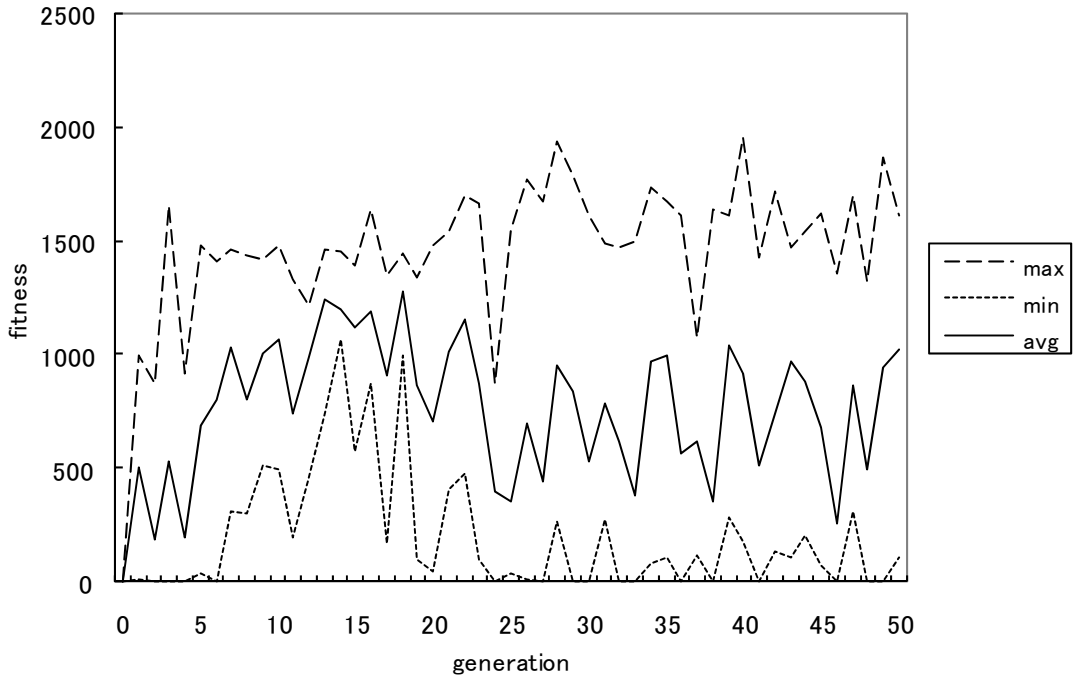


図 4-15　基本的な評価関数による進化過程

4.4.2. 淘汰圧を加えた評価関数による進化

壁に反応するためには，センサーからモータへのつながりがある回路構成を持った個体でないといけない．基本的な評価関数では，そういった個体が出現しても評価関数で評価されず淘汰されてしまっているようなので，センサー入力に反応する個体の評価値を上げるためモータが反転したら評価値が高くなるような淘汰圧を加えて実験を行った．

【評価関数】

$$F = D \times (1\text{-}S) + R \times A$$

[F:評価値　D:移動距離　S:センサ入力　R:モータ反転時にカウント　A:定数]

【結果】

<u>初期段階</u>

- 前進のみ，後進のみする
- その場で回転する

<u>30～50 世代</u>

- センサー入力に関係なくがたがたと前後する
- センサー入力に反応し壁の前まできて下がる

<u>100 世代～</u>

- センサー入力に反応し壁の前まできて前後する
- 衝突回避を行うが前後しながらよける

進化の初期段階では基本的な評価関数と同様に前後進のみや回転する個体も見られたが，10 世代を超えたあたりからセンサー入力に反応し，モータを反転させる個体が現れてきた．最終的には壁にぶつからずに行動する個体が大半を占めるようになった．しかしながら，壁に近づいて衝突回避を行う際，スムーズに回転して回避を行わずがたがたと前後進を繰り返しながら回避を行うような行動を獲得していた．これは，モータを反転すればするほど評価値が高くなるような評価関数となっているため，このような行動を獲得したものと思われる．

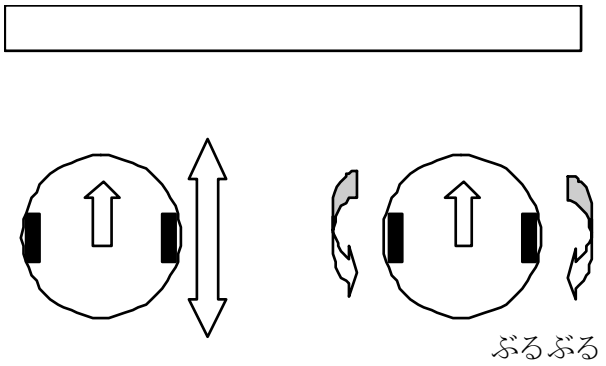


図 4-16　ロボットの動き

【進化過程】

基本的な評価関数と比べて非常に高い評価値を獲得できた．ちょうど，センサー入力に反応する個体が出現してきた 10 世代あたりから進化が進み 20 世代あたりで評価値が一定になりはじめている．

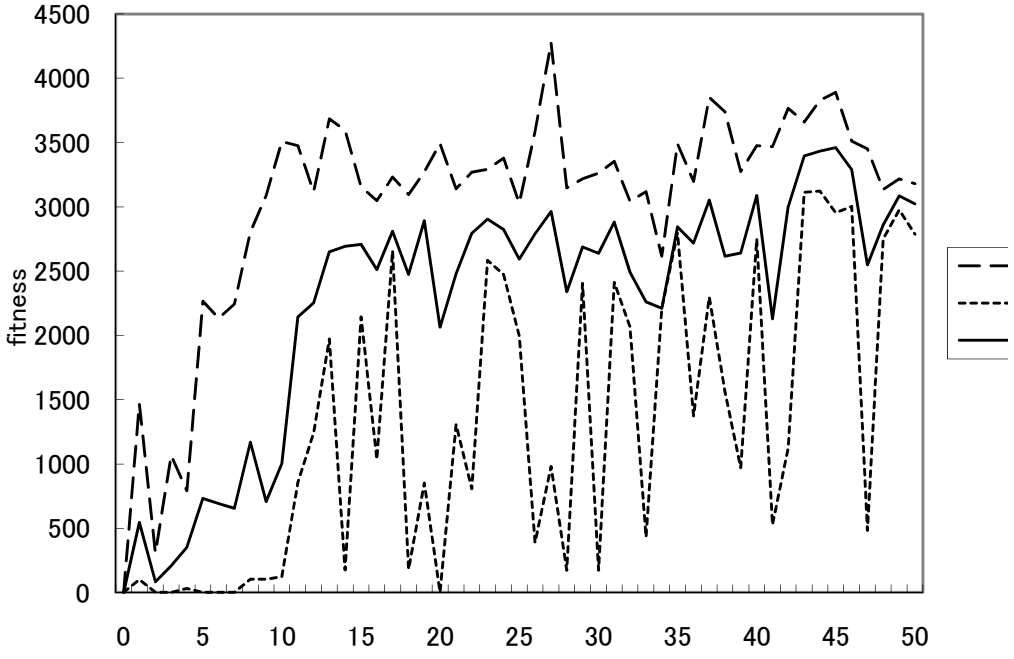


図 4-17　淘汰圧を加えた評価関数による進化過程

4.4.3. 進化過程での評価関数切り替え

淘汰圧を加えたまま進化させた場合，モータの反転により評価値が高くなるためがたがたとモータを前後させる行動を獲得した個体が支配するようになった．この行動を抑えるために，進化の初期段階でセンサーからモータへのつながりを得るために実験 2 の評価関数を使用し途中から実験 1 の評価関数に戻して実験を行なった．

【評価関数】

20 世代まで $F = D \times (1\text{-}S) + S \times A$

20 世代以降 $F = D \times (1\text{-}S)$

【結果】

初期段階

・前後進，回転する個体

30～50 世代

・センサー入力に反応し壁の前まできて下がる

・センサー入力に関係なくがたがたと前後する

100 世代～

・センサー入力に反応し壁の前まできて前後する

・衝突回避を行うが前後しながらよける

実験 2 の評価関数によって評価関数が切り替わる前に，高い評価値が得られる個体の割合が多くなったためか，評価関数が切り替わっても特に個体の動作としては変化が得られず，結果的に実験 2 と同様の結果となった．

4.4.4. 特定の論理演算器のみを用いた進化

これまでの実験では，論理演算器の機能も遺伝子によって変化するようになっていたが，この実験では演算器の機能を NAND のみに固定し実験を行なった．そのため，遺伝子情報から論理演算器の機能に関する情報を削除し，ひとつの遺伝子の長さは 12bit となっている．

遺伝子の構成を図に示す．また，論理演算器の機能分遺伝子の情報が減ったため 1 個体の遺伝子長は，5×5×12bit = 300bit となっている．

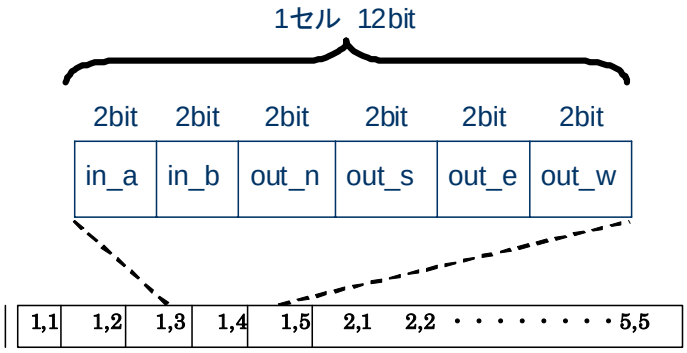


図 4-18　遺伝子の構造

評価関数は淘汰圧を加えた評価関数を使用した．

【評価関数】

$$F = D \times (1\text{-}S) + R \times A$$

[F:評価値　D:移動距離　S:センサ入力　R:モータ反転時にカウント　A:定数]

【結果】

初期段階

- 前進のみ，後進のみする
- その場で回転する
- センサー入力に反応してモーターを前後させる

30～50 世代

- センサー入力に関係なくがたがたと前後する
- センサー入力に反応し壁の前まできて下がる

行動としては淘汰圧を加えた評価関数による進化と同様の行動が得られたが，センサー入力に反応する個体が進化のかなり早い段階で見られた．

【進化過程】

これまでの実験と比べて非常に早い段階で高い評価値を獲得している．論理演算器の機能の選択のための 4bit がなくなり，1 個体では 100bit の遺伝情報がなくなったため遺伝子が短くなり進化も速く進んだと考えられる．

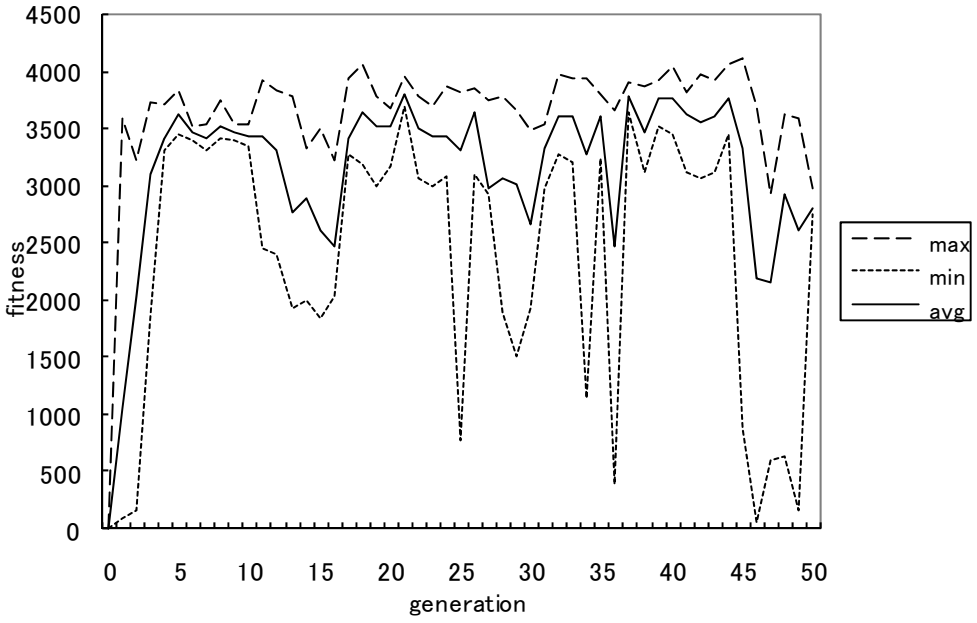


図 4-19　実験 4 の進化過程

4.4.5. Microbial GA による進化

遺伝子を進化させる GA として Microbial Genetic Algorithms を使用した実験を行なった．利用した評価関数は淘汰圧を加えた評価関数を使用した．

【評価関数】

$$F = D \times (1\text{-}S) + R \times A$$

[F:評価値　D:移動距離　S:センサ入力　R:モータ反転時にカウント　A:定数]

【結果】

初期段階

・ほとんどが前後進のみ

30～50 世代

・センサー入力とは関係なくモーターを反転させた動きを行なう

・センサー入力に反応して壁の前まできて下がる

進化の初期段階ではこれまでの実験と同様に前後進のみを行なう個体が多く，回転する個体も数個体みられた．20 世代を超えたあたりからセンサー入力に反応する個体も出てきたが，観察していてわかるほどの進化は見られなかった．

【進化過程】

最終的な評価値は基本的な評価関数による進化実験とほぼ同じ値程度となった．しかし，基本的な評価関数による進化ではすぐに進化が止まっているのに対して，この実験では少しずつではあるが着実に進化して行っているのがわかる．

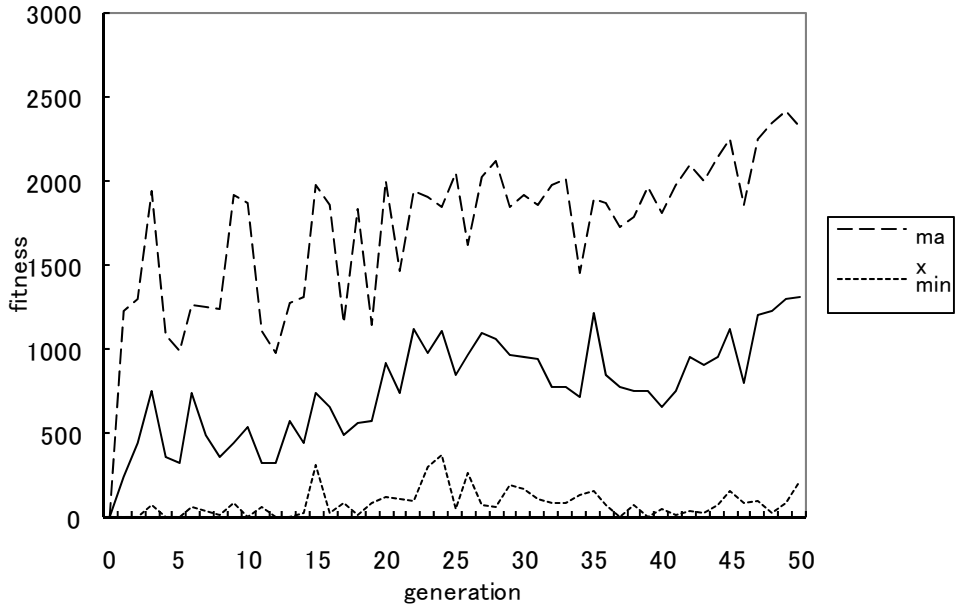


図 4-20　MicrobialGA による進化過程

4.5. 考察

4.5.1. 基本的な評価関数による進化

基本的な評価関数では FPGA の回路構造を淘汰していくほどの淘汰圧がかからずに，ほとんど評価関数として機能していないといえる．また，実際の回路を移動ロボットの制御コントローラとして利用しているため，センサー入力からモータ出力まで回路的につながった構造になっていなければ，壁を回避することは不可能であるが，そのような構造をもった個体はほとんど見られなかった．

4.5.2. 淘汰圧を加えた評価関数による進化

ここでは，基本的な評価関数による進化実験の結果を踏まえ，てつながりを持つ回路構成を持った個体が残るように，評価関数を改良した．モータの回転が変化する要素としては，回路的に発信回路または出力があるタイミングで切り替わるような回路が内部的に発生している場合か，もしくはセンサー入力の信号がモータ出力に栄了した場合である．壁との衝突回避を行なうためには，モータの回転方向を変えてロボットの向きを調整する必要があり，モータが反転動作したときにカウントして定数をかけたものを加える評価関数を使用した．この方法では，センサー入力からモータ出力までつながりを持った回路もしくは，何らかの要因でモータ出力が変化する回路が構成されるはずである．この評価関数を使った実験から得られた結果では，壁の前までくるとモータを反転させる行動を行う個体が発生し評価関数が有効に機能していることがわかった．しかしながら，壁の前まで来て前後に移動する行動を行なう個体が大半を占め，壁を回避する行動を行った個体はごくわずかであった．これはセンサーの入力数やモータへの出力数，使用しているセルの数などの要因によって，壁を感知して徐々によけるといった複雑な制御モデルを構成できなかったためだと考えられる．

4.5.3. 進化過程での評価関数の切り替え

ここでは，淘汰圧を加えた評価関数による進化実験でのがたがたと前後に動く行動を極

力残さないために，進化の最初だけ淘汰圧を加えてセンサー入力からモータ出力までのつながりを持った個体を残すようにした．しかし，淘汰圧を加えている間にがたがたと前後に動く行動を獲得してしまい，この後の進化過程でもその行動は残ってしまった．このことから，モータが反転したら評価値を上げるという評価関数そのものに問題があることがわかった．

4.5.4. 特定の論理演算器のみを用いた進化

これまでの実験では論理演算器の機能を遺伝子によって自由に決定できるようになっていたが，論理演算器の機能が変わることによって行動に影響するかどうかを調べるために，論理演算器の機能を NAND だけに固定し実験を行なった．得られた行動としては，淘汰圧を加えた評価関数による進化と同様であったが，非常に早い段階で進化が行なわれた．淘汰圧を加えた評価関数による進化では 10～15 世代あたりから進化が急速に進んでいるが，この実験では 5 世代に達するまでに高い評価値を得るまでに進化した．これまでの実験では 1 個体の遺伝子長が 400bit あったが，この実験では 300bit と 3/4 の長さに短くなったため，探索空間も狭くなりより早く最適な解を発見できるようになったためと考えられる．

4.5.5. Microbial GA による進化

実験 5 では，基本的に実験 2 と同じ条件で，GA だけを Microbial Genetic Algorithms を使用して実験を行なった．同じ世代数では実験 2 よりも評価値が低く，実験 1 と差ほど変わらない結果となってしまった．しかし，進化過程のグラフを見ればわかるとおり，実験 1 ではすぐに評価値が飽和し進化が止まってしまっているが，実験 5 ではゆるやかではあるが評価値が上がり続け進化していることが確認できる．このまま進化を続けていけば，実験 2 と同等かそれ以上の評価値を得ることも可能かもしれない．

4.6. 今後の課題

今回5×5セルと非常に少ないセル数で実験を行ったために行動の制限が出た可能性も考えられる．また，khepara から利用できる FPGA の入出力ポートの位置の都合上，センサー入力からモーター出力までが一定の距離ではなく，短いところではセンサーからモーターまでが 2 つのセルしかない場所もあったため，入出力の場所の影響もあったと考えられる．そのため，センサー入力からモータ出力まで回路全体を使えるような構成に変更し，また使用するセルの数を増やして実験してみる必要がある. また, センサー入力を閾値で 0, 1 として入力を行っているため環境からの情報が非常に少ないといえる．そのため，センサー1 つにつき 2bit～4bit ぐらいの情報を持たせて FPGA に入力することにより，障害物の予測をする機能を獲得する可能性もある．同様に，モータも 0，1 による前後の動きだけであったが，これも情報量をふやすことで停止状態やスピードのコントロールなどもできるような自由度を与えてやると，よりスムーズな動きを獲得する可能性は高い.

今回の実験から明らかになったが，FPGA を移動ロボットの制御コントローラとして利用する場合は，センサー入力からモータ出力までの回路的なつながりを維持しなければ，役に立たないことがわかった．そのため，評価関数だけではなく，必ずセンサー入力からモータ出力までの回路的なつながりを維持するような仕組みを回路構成に持たせたほうが良いと思われる．この仕組みを作ることで，ある程度のモデルを与えてしまうことになるが，移動ロボットの場合，センサー入力に対する反応がないことには意味をなさないので有効な方法と考える.

また，Microbial Genetic Algorithms を使った実験では，今回実験を停止した世代まで進化し続けていたため，この後の世代でも進化していくことが十分に考えられる．従来型 GA と同条件で高世代まで進化させてみて，その進化過程を比較する必要がある.

4.7. まとめ

今回の基礎的実験により，自律行動型ロボットの制御コントローラとして進化型ハードウエアが環境に適応して進化的に制御構造を獲得することが確認できた．今回の実験では

スムーズな衝突回避までは獲得できなかったものの壁にぶつからない制御構造を獲得できた．また，入出力ポートの配置の関係で利用可能なセルを十分に使用していない可能性が高く，実際のハードウエアにすると少ない部品点数で実現できているといえる．

進化型ハードウエアで所望の動作を獲得するには評価関数の設定が非常に重要になってくるものと思われる．また，進化型ハードウエアは制御構造自体を進化的に獲得していくために，進化の初期段階では制御構造自体が構成されていない場合が多い．特に自律行動型ロボットの場合，センサー入力に対してモータが反応するといったことが重要な要素になってくるため，センサー入力からモータ出力までのつながりを維持するような回路的な構成が必要に思われる．

本研究では，遺伝操作の新しい手法として Microbial Genetic Algorithms を実際にロボットに適用する試みを行なったが，着実に進化していくことを確認できた．今回の実験では 50 世代程度までしか確認できなかったが，より高い世代まで進化させることによって非常に興味深い結果が得られるのではないかと考えられる．

4.8. この章の参考文献

[1] R.A.Brooks: A Robust Layered Control System for a Mobile Robot, RA-2, No.1, pp.14-23, Mar.1986

[2] 五味 隆志: 非デカルト的知能ロボット

[3] R.A.Brroks, 五味 隆志: 複数の要素行動間の競合・強調により知能ロボットの行動を決める「サブサンプション・アーキテクチャー」，日経インテリジェントシステム別冊春号 pp.152-167

[4] 北野 宏明: 遺伝的アルゴリズム，産業出版

[5] 北野 宏明: 遺伝的アルゴリズム 2，産業出版

[6] 木村 元，宮崎 和光，三上 貞芳，皆川 雅章，三上 敬，高取 則彦，鈴木 恵二共訳: 遺伝アルゴリズムハンドブック，森北出版株式会社

[7] Inman Harvey: Artificial Evolution: A Continuing SAGA , ER2001 LNCS2217 pp.94-109,2001

[8] Inman Harvey: The Microbial Genetic Algorithm, 1997

[9] Inman Harvey: Evolutionary robotics and SAGA: the case for hill crawling and tournament selection, Appears in C. Langton (ed.), Artificial Life III, Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XVI, pp. 299--326. Addison Wesley 1994.

[10] Adrian Thompson: An evolved Circuit, Intrinsic in Silicon, Entwined with Physics, 1st Int. Conf. on Evolvable Systems 1996

[11] Adrian Thompson: Silicon Evolution. Genetic Programming 1996.

[12] Yong Liu, Tetsuya Higuchi, and Masaya Iwata: Design of Evolvable Hardware for Robotic Navigation, Wuhan University Journal of Natural Sciences , vol. 6, no.1-2, pp. 547-554,2001.

[13] T. Higuchi, H. Iba, and B. Manderick: Evolvable Hardware, Massively Parallel Artifical Intelligence, pp.398-421, MIT Press, 1994.

[14] I. Kajitani, T. Hoshino, M. Iwata, and T. Higuchi: Variable Length Chromosome GA for Evolvable Hardware, Proc. of 3rd Int. Conf. on Evolutionary Computation (ICEC96), Nagoya,1996.

5. 自律移動型ロボットのための集合ニューラルネットワークの進化

5.1. はじめに

自律移動型ロボットの制御システムを人工的に進化させる研究が広く行われている[1, 2, 3, 4, 5]．自律行動型ロボットにおける制御コントローラーの人工的進化は，ロボットの制御構造や形態などの各種パラメータをコード化して染色体として表現し，数個の染色体（個体）で世代を形成する．すべての個体は目的とする行動の優劣をつけるために，評価関数によって選択を行い，遺伝オペレーターによって新しい世代の作成を行う．そして，評価基準が満たされるまで，古い世代から新しい世代を生成し進化させていく．また，ロボット工学における進化アルゴリズムとファジー理論の組み合わせ[6, 7]や，進化アルゴリズムと学習による相乗効果の研究が行われており関心を得ている[8, 9, 10, 11]．進化と学習は，異なったタイムスケールで生物学的な適合を行うもので，進化は比較的遅い環境の変化をとらえて適合を行っていくが，学習は各個体ごとで行われるため遺伝的アルゴリズムにはない適応の変化を発生させる可能性がある．また，学習を組み合わせることによって，遺伝的アルゴリズムの進化過程のスピードが速くなるという報告もある．

自律行動型ロボットの制御コントローラーとして，フィードフォワードネットワークや再帰的ネットワークなど多種多様なニューラルネットワークの構造と進化のアプローチが試みられている．それらのネットワークは有用であるが，隠れ層のあるなしにかかわらず単一のニューラルネットワーク（Single Neural Network : SNN）で構成されている．これらの研究の多くは，与えられたタスクが与えられた環境における最適解を見つけられる条件が整っていたり，ネットワーク構造もしくは機能が設計者のアプリオリ的な知識に基づいて作成がおこなわれている場合が多い．しかしながら，Harvey[19]によって提起されていうように，実際のオープンな環境における最適解の探索範囲のなかには，同じような最適解が複数存在している．オープンな環境で動作する自律行動型ロボットは，特定の状況に応じて最も適切なふるまいを発生させる能力を持つべきである．

これは言い換えれば，学習と記憶における適応と安定の問題と考えられる．ロボットが特定の状況を感知し対応する動作を行う能力を取得した時に，その時の動作パターンを記

憶しておき，わずかな状態変化に応じて対応行動パターンが変化した場合，ロボットは前の行動パターンは消さずに，その時の対応行動パターンを記憶し適応するのが理想である．しかしながら，従来の進化や学習では，ロボットは新しい行動パターンの中に過去の行動パターンを一般化してしまい，過去の記憶を消してしまう場合がある．また，遺伝オペレーションの中でより高い評価値を得た個体が勝者となり，ロボットにとって重要な個体であってもわずかな評価値の違いにより敗者は淘汰されてしまう．

近年，集合ニューラルネットワーク（Ensemble Neural Networks：ENNs）は，多くの現実的な問題を解決するために研究されている[17]. 集合ニューラルネットワークは数個の単一ニューラルネットワークの組み合わせであり，単一ニューラルネットワークより有用性が示された[18]. 集合ニューラルネットワークは多くの現実的な問題で利用されているが，自律行動型ロボットの制御システムのために集合ニューラルネットワークを適用した例は非常に少ない．集合ニューラルネットワークの研究の中に，数個のコンポーネントネットワークをスイッチングする方法が提案されている[20]．しかしながら，希望する目的を満たすためには多くのスイッチングルールを記述する必要があり，適応するルールの生成は難しく，スイッチングルールを設計するのが厄介な問題と言わざるを得ない．コンポーネントネットワーク間同士の協調が，入力センサーや出力値による学習や遺伝的アルゴリズムによる進化によって得られれば，前途のような問題が解決するといえる．

これらのことより，自律行動型ロボットの制御システムとして，集合ニューラルネットワークはロボットが現実的に遭遇する問題に対して，従来のコントロールシステムより可能性があるのではないかと考えた．本研究では，集合ニューラルネットワークを自律行動型ロボットに適用し自律的構造化する試みを行った．コンポーネントネットワーク間を進化と協調により結合するようなコントローラーの設計行った．この手法では，各世代におけるすべての個体は評価関数によって評価され遺伝的オペレーションによって進化を行い，またすべての個体は各コンポーネントネットワークが協力を行うための協調機能を学習する．協調学習の目的は，ロボットの新しい技量や才能の獲得である．このような，協調的にロボットの機能を得る利点として，ロボットが活動を行う環境に関して何も仮定する必要はなく，各コンポーネントネットワークが自律的に環境の情報とそれに対応する構造を得るということである．

5.2. 集合ニューラルネットワークの進化

5.2.1. 進化と協調学習の実装

協調を行う複数のニューラルネットワークの中に，同じ振る舞いをするニューラルネットワークが存在すると意味がないため，同様の振る舞いを行わないよう相反する相関関係を取り入れた．これにより，特定のニューラルネットワークが特定の仕事を行うことが予想される．各コンポーネントネットワーク間で相反する相関関係学習を行うため，それぞれのコンポーネントネットワークの出力を関連させる必要がある．しかしながら，Liu と Yao(1999)[22]によって提案された協調ネットワークにおける相反する相関関係学習は，そのままロボットの制御コントローラとして利用するには不向きである．本研究では，事前に学習データを取得するのは目的にそぐわないと考え，その代わりに協調機能が最小限になるよう行った．

M と $F_i(n)$はそれぞれ，コンポーネントネットワークの総数と入力 N のサンプルのネットワーク L の出力とする．出力の平均を $F(n)$とすると次式で表される．

$$F(n)=\frac{1}{M}\sum_{i=1}^{M}F_i(n) \tag{5-1}$$

Liu と Yao によって提案されたものと同じ相関関数を最小にすることで入力サンプル n 個のとき i 個のネットワークは $p_i(n)$として定義される．

$$p_i(n)=\tfrac{1}{2}\left(F_i(n)-F(n)\right)\sum_{j=1,j\neq i}^{M}\left(F_j(n)-F(n)\right) \tag{5-2}$$

n 番目の入力サンプルのネットワーク i の出力に関する Pi(n)の偏導関数は

$$\frac{\partial p_i(n)}{\partial F_i(n)}=\sum_{j=1,J\neq i}^{M}\left(F_j(n)-F(n)\right) \tag{5-3}$$

$$=-\left(F_i(n)-F(n)\right) \tag{5-4}$$

したがって，荷重の変化量の更新ルールは下記のとおりである．

$$\Delta w_i(n) = -\frac{\partial P_i(n)}{\partial w_i} = -\frac{\partial p_i(n)}{\partial F_i(n)}\frac{\partial F_i(n)}{\partial w_i} \tag{5-5}$$

$w_i(n)$ と $\Delta w_i(n)$ が，それぞれ i 番目のコンポーネントネットワークの荷重と，n 番目の入力サンプルでの荷重の変化量である．各コンポーネントネットワークの出力は，L 次元でのセンサー入力 x_l と結合加重 w_l の総和になるとみなし次式で表す．

$$F_i(n) = \sum_{l=1}^{L} w_l(n) x_l$$

結合加重の差分を Δw_i，t 回繰り返した後の i 番目のコンポーネントネットワークの結合加重を w_i^t とすると以下の通りになる．

$$\Delta w_i(n) = \left(F_i(n) - F(n)\right)x_i \tag{5-6}$$

$$w_i^t = w_i^{t-1} + \frac{\lambda}{N}\sum_{n=1}^{N}\Delta w_i(n) \tag{5-7}$$

パラメータ $\lambda > 0$ は相関関係学習の強さ，n は繰り返し利用される入出力データである．本研究で用いたロボットのセンサー入力は 0 から 1023 の値をとり，ネットワークの出力もおおよそ 0 から 1023 の値をとるため，計算を容易にするため 0 から 15 の値をとるように変換を行った．また，本実験は N=10 として行った．

自律行動型ロボットが意味のある協調動作を行うには多くコンポーネントネットワークを必要とするかもしれないが，提案したアルゴリズムではどれだけ多くのコンポーネントネットワークが存在しても有効であると考えられる．本実験では，2 層の階層型ニューラルネットワークをコンポーネントネットワークとし，2 つ，3 つ，4 つの場合でコンポーネントネットワークを協調学習させた．

5.2.2. 集合ニューラルネットワークの手順

実際の移動ロボットに, 集合ニューラルネットワークを適用した場合のプロセスを図 5-1 のフローチャートで示す． まず，1 個体あたり 5 秒の生存期間で環境内を移動させて全ての個体が終了した後，評価値の評価を行う．次に先ほど得られた各個体を同じ生存期間で移動させる．今回，遺伝的オペレータは適用せずに，代わりに協調関数を適合させるための 10 セットの入出力データを収集した．以上の過程を必要な世代数まで繰り返す．この実験手順の詳細については，この後のセクションにて説明を行う．

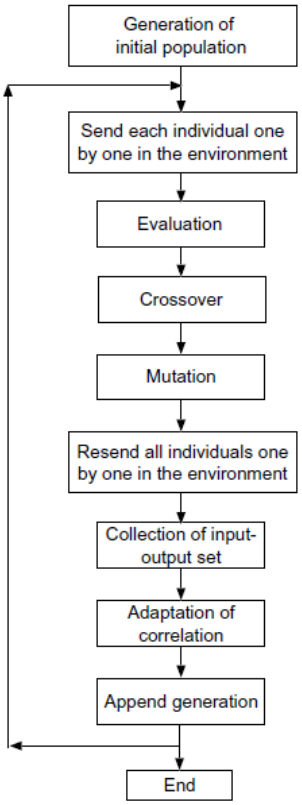


図 5-1 集合ニューラルネットワークの進化手順

5.3. 実験内容

5.3.1. 実験環境

本研究では，実際の移動ロボットとして 4.3.1 で説明した小型移動ロボット Khepera を使用した．ロボットと PC の接続は，図 5-2 の上図に示すように，ロボットの RS232C ポ

ートと Linux を搭載した PC のシリアルポートを，ローターリーコネクターを介して接続している．ニューラルネットワークの構築と遺伝的操作の実行は，外部の PC で処理を行った．また，センサー入力に対するモーター出力の処理と，センサー値およびモーターに接続されたエンコーダーのモニターは，ロボットに搭載されているマイクロプロセッサによって処理を行った．進化させた環境は，図 5-2 の上図に示すように，縦 60×横 60cm の木製の板で囲った大きな正方形の中に小さな 4 つの正方形で障害物をつくりループ状の環境を作成した．外周の板および中の障害物には，センサーに影響がないように光沢の無い段ボール紙を張った．実験環境は，外部からの光が入らないよう全体を囲っており，上部から蛍光灯 2 本と 60 ワットの白熱球によって照らすようになっている．

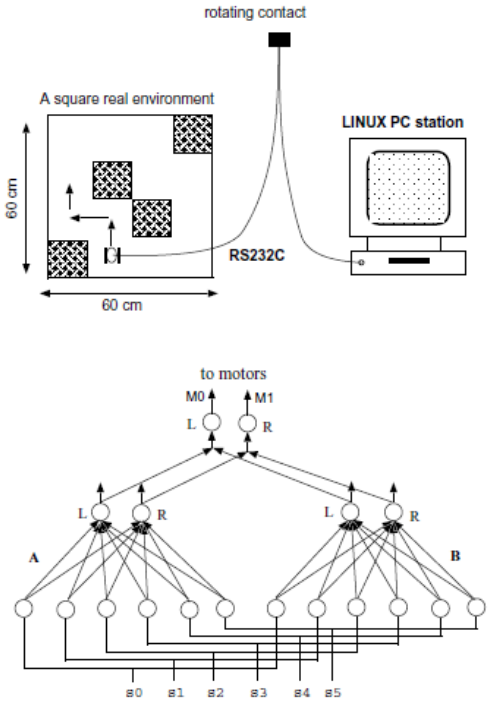


図 5-2 実験環境（上図）とニューラルネットワークの構造（下図）

5.3.2. 制御ネットワークと遺伝子構成

シンプルな 2 層で構成したニューラルネットワークを，集合ニューラルネットワークのコンポーネントネットワークとして使用した．2～4 つのコンポーネントネットワークをモーターの制御信号を作り出すための制御システムとして使用している．例として，コンポ

ーネントネットワークが 2 つの場合を図 5-2 の下図に示す．モーターを制御する 2 つの制御信号は，赤外線センサーやビジョンセンサーからの値を積算したものによって生成する．

$$S'_m = S_b + G\sum_{l=1}^{L} w_l x_l \qquad (5\text{-}8)$$

$$S_m = \begin{cases} +10 & S'_m > 10 \\ S'_m & -10 \le S'_m \le 10 \\ -10 & S'_m < -10 \end{cases}$$

Sm，S'm，Sb，G，Wl，Xl はそれぞれ，モーターへの出力値，モーターへの出力の閾値，モーターの基本移動スピード，全体のゲイン，結合荷重，センサーの入力値となっている．Sb， G， L， の値はそれぞれ，5，1/1600，6 に設定した．全体のゲインは，センサーからの入力信号の感度の調整を行う．最後のモーターへの出力信号は，コンポーネントネットワークの出力の平均値から生成を行った．例えば，2 つのコンポーネントネットワークの左側の出力の平均値を左側のモーターに，同じように 2 つのコンポーネントネットワークの右側の出力の平均値を右側のモーターに利用した．

集合ニューラルネットワーク全体の結合加重をコード化し遺伝子とした．それぞれの荷重を 5 ビットの遺伝子で構成しており，最初の 1 ビットで荷重の符号を表し残り 4 ビットで強さを表している．本研究ではロボットの前方 6 個のセンサーを用いたため，コンポーネントネットワークの荷重は全部で 12 個となる．したがって，2 つのコンポーネントネットワークの必要な遺伝子長は 5×12×2=120 bits となる．

ロボットは近接センサーで障害物などとの距離を知ることができ，その入力センサーのレンジは 0 から 1023 までの値で取得することが可能となっている．図 5-3 にセンサー入力値の例を示す．この例では，簡単な障害物回避行動を行っているロボットのセンサー番号 3（上図）と 4（下図）のもので，0.1 秒間隔でサンプリングを行っている．

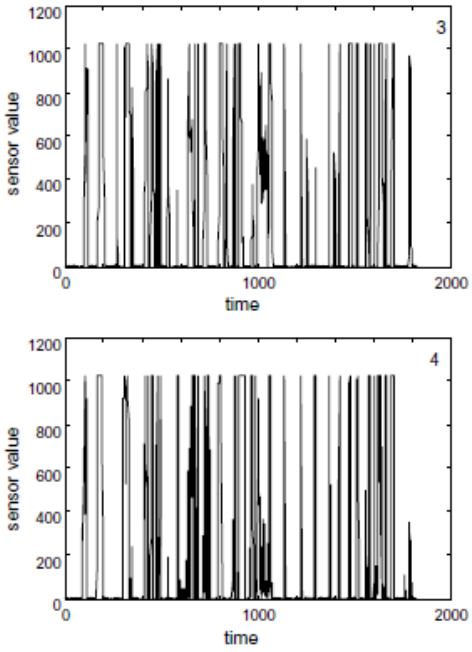


図 5-3 実験環境からのセンサー入力例

5.3.3. 評価関数とタスク

ロボットが行うタスクとして障害物を避けながら移動することとした．進化と相関関係学習は連続して行っており，全ての世代における全ての個体を実験環境で決められた時間移動させている．本研究で用いた評価関数は，以下で示す広く用いられているものを利用した．この評価関数は，移動速度をなるべく落とさずスムーズに障害物回避行動を行うものとなっている[24].

$$f = \sum_{t} V(t)(1-\Delta v(t))\left(1-\sum_{i=1}^{n} s_i(t)\right) \qquad (5\text{-}9)$$

V は 2 個のモーターの平均回転速度で移動速度が速くなるとよい評価となる．Δv はモーターの速度の絶対値の差分で直線的な移動により評価値が高くなるようになっている．s_i は，i 番目の近接センサーの値で，ロボットが障害物に近づくほど値が高くなり評価値が低くなるようになっている．ここで，Δv と $\sum s_i$ の標準化を行ったので，取りうる最大値は 1 と

なっている.

5.3.4. 遺伝的アルゴリズムのパラメータ

基本的な遺伝的アルゴリズムを，ニューラルネットワークにおける結合荷重を決定するのに利用した．本研究で利用した遺伝的アルゴリズムのパラメータを表 5-1 に示す．進化の間，Braitenberg アルゴリズム[25]によると，ロボットは前の世代の個体の影響を避けるために位置をランダムに変更する.

表 5-1 Genetic parameters used for evolution

Population size	10
Number of generation	20-570
Cross-over probability	0.2-0.5
Mutation probability	5 bit
Elite preservation	2
Bit per weight	5 bit
Chromosome size	120 bit
Final weight range	0-15
Life time (indiv)	5 sec
Sample rate	100 ms

5.3.5. 実験手順

ロボットが移動している間，ロボットと環境との相互作用は重要な問題である．ロボットはセンサーからの情報入力により環境の情報を集める．センサー情報をコントローラーに入力し，その出力によってモーターを駆動してロボットを移動させる．進化と学習は，まず，すべての個体に遺伝的オペレーションを行い進化させ，その次に各個体個別に協調学習を行った.

第 1 段階

まず最初に，外部のワークステーションにて遺伝子（個体）を生成し，数個のまとまった個体を 1 世代としてロボットに搭載させれているマイクロプロセッサへ転送を行う．そして，遺伝子の情報をニューラルネットワークに適用し，0.1 秒毎にセンサー入力をニューラルネットワークに入力し，その出力によってロボットのモーターを駆動して移動を行った．移動した距離はモーターに付属しているエンコーダーによって収集し，各個体の評価のためにワークステーションへ転送を行った．以上の手順を全ての個体に対して行いデータの収集が終わったのち，選択，交叉，突然変異の 3 種類の遺伝的オペーレーションを行い，次の世代を生成した．

第 2 段階

第 1 段階の後，協調学習を行うために同じ手順を行った．進化と協調学習を同時に行うには，ロボットの動作とワークステーションでの処理時間に問題があったため，別に処理を行うこととした．第 1 段階でロボットのマイクロプロセッサに転送済みの遺伝子を使って，環境内でロボットを移動させた．第 2 段階では，遺伝的オペレーションは行わないため，ロボットは 10 セットの遺伝子を使ってセンサー入力に対応する出力によってモーターを駆動させ移動距離を収集した．すべての個体の試行が終わった後，収集した移動データは協調学習を行うためにワークステーションへ転送を行った．そして，式 5-7 によって協調学習を行いすべての個体の結合荷重の更新を行った．この第 1 段階，第 2 段階の処理を経て世代更新を行った．

5.4. 実験結果

実験は静的環境と動的環境の 2 種類で行った．静的環境では，温度，光度，数個の光源など環境条件が固定されている．一方，動的環境ではそれらの環境条件を可変して実験を行った．

5.4.1. 静的環境における進化と学習過程

(1)進化過程

図 5-4 に，静的環境における協調学習を行う集合ニューラルネットワークの進化過程を示す．1 世代 10 個体で進化を行い，世代における評価値の平均値をグラフ化している．進化が進むにつれて評価値の向上が確認できた．ロボットは，30～40 世代で障害物回避行動の獲得を行っていた．

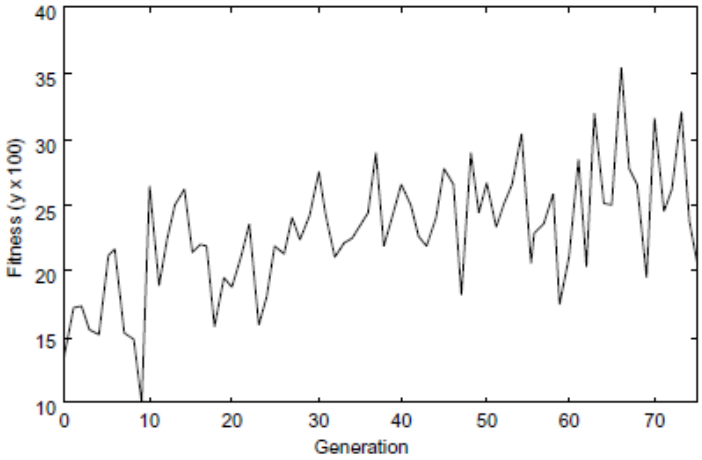


図 5-4 静的環境における平均評価値

(2) 相関関係

各世代の進化後の，右モーター用の 2 つの右側出力と左モータ用の 2 つの左側出力の相関関係を図 5-5 に示す. それぞれのコンポーネントネットワークのために 10 個体の出力を収集し相関関係の計算を行った．そして，各世代の進化後，10 個体のための 10 個の相関関係の平均値をプロットした．進化の初期段階において相関関係はばらばらで，正と負の値が入り乱れていたが，進化が進むにつれて負の相関関係になったことがわかる．このように荷重を少しずつ更新した場合，たがいに負の相関関係になるまで少し時間を要する．右モーター用の 2 つの右側出力と左モータ用の 2 つの左側出力の 1 対の相関関係を式 5-10 によって計算した.

$$C_{AB} = 100 \times \frac{\sum_{t=1}^{T} \left(F_A(t) - \overline{F}_A\right)\left(F_B(t) - \overline{F}_B\right)}{\sqrt{\sum_{t=1}^{T} \left(F_A(t) - \overline{F}_A\right)^2 \sum_{t=1}^{T} \left(F_B(t) - \overline{F}_B\right)^2}} \quad (5\text{-}10)$$

C_{AB} はネットワーク A と B.間の相関関係である．$F_A(t)$ と $\overline{F}_A$ はそれぞれ，ネットワーク A のための入力サンプルもしくはパターンとネットワーク A の全出力の平均値である．

そして，各世代の進化後のすべての個体の相関関係の総和をとった．

$$C_{sum} = \frac{1}{10} \sum_{i=1}^{10} C_{AB}^{i} \quad (5\text{-}11)$$

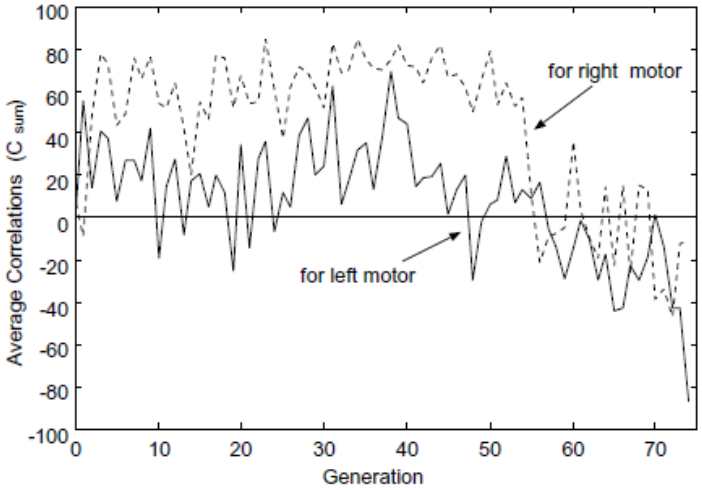


図 5-5 各モーターのための各出力の時間相関関係

(3) 結合荷重の機能

協調機能の影響を確認するために，結合荷重の分配の記録を行った．図 5-6 に 2 つのコンポーネントネットワークを用いた場合の結合荷重の値を示す．1～6 と 7～12 は，センサー入力 0～5 と右と左の出力ユニットとの結合荷重を示す．2 つのコンポーネントネットワークの結合荷重は，明らかに機能的違いを示した．初期の世代，50 世代目，75 世代目の状態を図示している．進化を行う初期の世代から 50 世代目までは，結合荷重からは 2 つのコンポーネントネットワーク間での機能的違いは見られなかったが，75 世代目になると結合

荷重は両方のネットワークがそれぞれ異なった機能を有することを示した．このとき 2 つのコンポーネントネットワークは負の相関関係となっている．

この学習の本質は，ロボットの動きが 2 つの補完的な機能によって実現できている点である．このあと，違った行動をロボットが取る可能性はあるが，単一ニューラルネットワークではこのような動作は不可能である．

2 つのコンポーネントネットワークの収束した結合荷重は，ほぼ$W_A = (w_1, -w_2)$ と$W_B = (-w_1, w_2)$になっていたが，いくつかの結合荷重は同じ符号となっていた．相反する協調学習を行ったため，このような結合荷重の値をとるのは理想的であるといえる．この場合，2 つのコンポーネントネットワーク間ではほとんど同一符号の結合荷重にはならないと思われる．これらは，集合ニューラルネットワークが協調学習だけでなく進化による淘汰圧も影響していると考える．したがって，2 つのコンポーネントネットワークはお互いを完全に補完的しないと思われる．

図 5-7 に 3 つのコンポーネントネットワークを使用した場合の結合荷重の値を示す．2 つのコンポーネントネットワークの場合と同様に，後半の世代においてコンポーネントネットワーク間の結合荷重が補完的となった．図 5-8 は 4 つのコンポーネントネットワークの場合の，100 世代目の一番良い個体の結合荷重を示している．この場合も各コンポーネントネットワークの結合荷重が補完的になっているといえる．

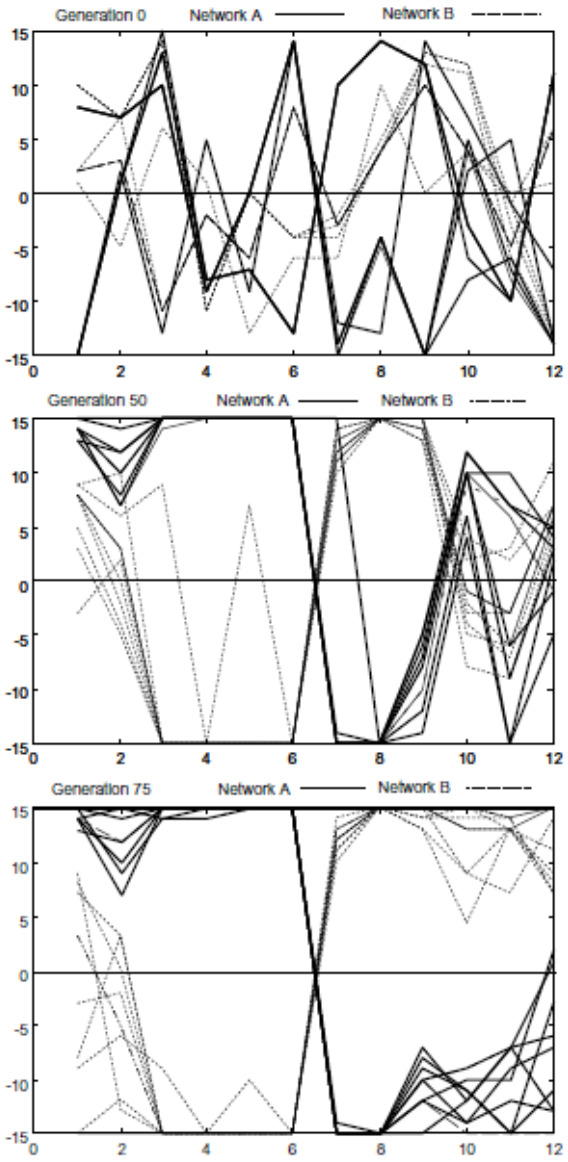


図 5-6　2 つのコンポーネントネットワークの場合の静的環境における結合荷重

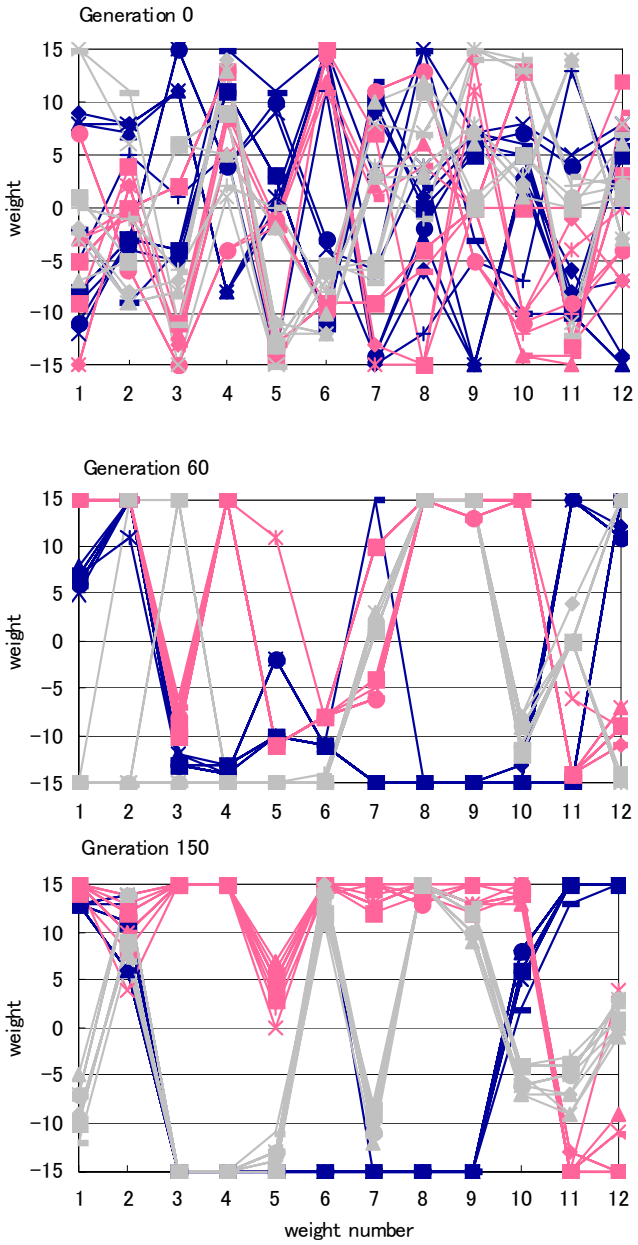


図 5-7　3 つのコンポーネントネットワークの場合の静的環境における結合荷重

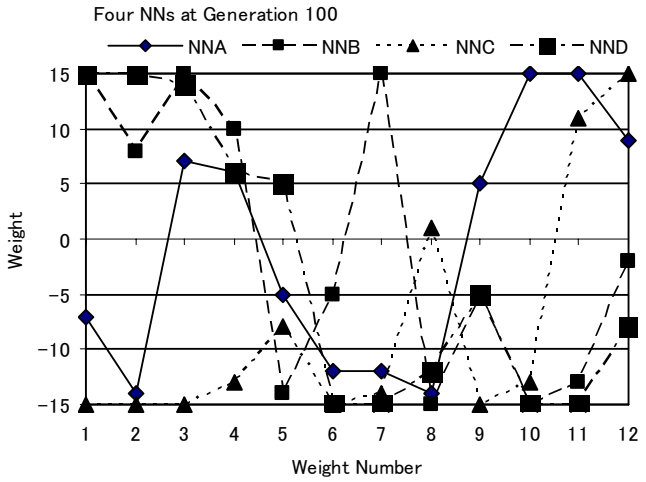


図 5-8　4 つのコンポーネントネットワークの場合

(4) 結合荷重の相違

3 つのコンポーネントネットワークを使用しているとき, 以下の方程式でコンポーネントネットワークの結合荷重の相違を測定できる.

$$D = \sqrt{\left(\left|W_A - W_B\right|^2 + \left|W_B - W_C\right|^2 + \left|W_C - W_A\right|^2\right)/3}$$

W_A, W_B, W_C はそれぞれ下記のような結合荷重のベクトルを示す.

$$W_A = (w_{A1}, w_{A2}, ..., w_{A12}), W_B = (w_{B1}, w_{B2}, ..., w_{B12}), W_C = (w_{C1}, w_{C2}, ..., w_{C12})$$

図 5-9 に 2 から 4 つのコンポーネントネットワークのそれぞれの進化過程を示す. コンポーネントネットワークが 2 つ，3 つ，4 つの場合でのそれぞれの違いは，図からわかる通り

後半の世代で現われてくる．

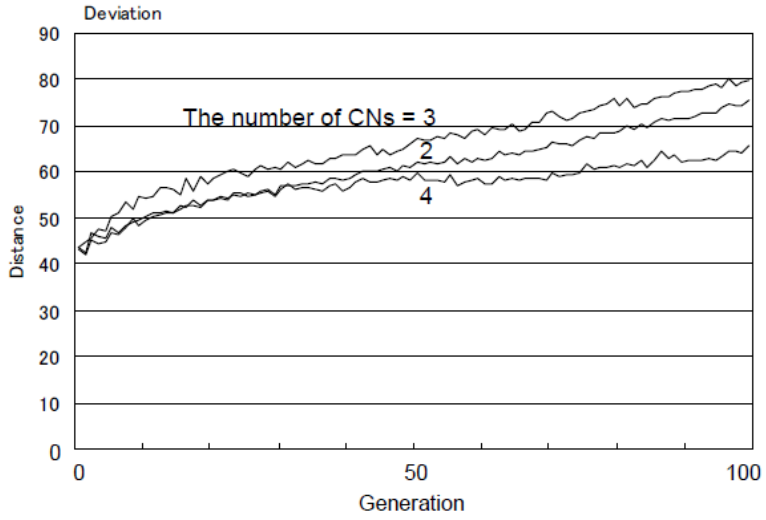


図 5-9 コンポーネントネットワーク間の相違

5.4.2. 動的環境における進化と学習過程

(1) 進化過程

動的環境は，60 ワット電球の光量と位置の変更と 2×3cm の長方形の障害物を置くことによって作成した．300 世代の進化を行うのにおよそ 13.5 時間必要となる．30 分毎に光量を 3 段階で切り替えた．全ての切り替えが終わった後，電球 L の位置を図 5-10(a)のように移動させた． 長方形の障害物は，100 世代ごとに 30 分間だけ環境内に設置した．

図 5-11 に図 5-10(a)の環境下で進化させたときの評価値を示す．進化が進むにつれて評価値の向上がみられるが，60～70 世代と 160～170 世代のあたりで評価値の低下がみられる．これは，ロボットが行動している環境に変化があったためである．ロボットは移動中，電球の調整のためにノイズを受信していたが，衝突回避行動を継続して行っていた．

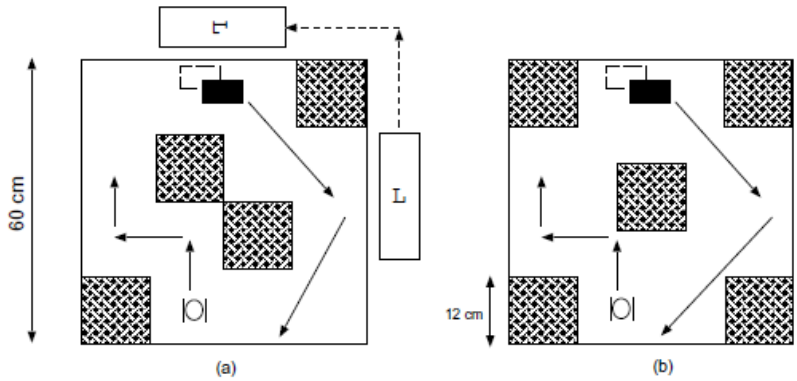


図 5-10 動的環境

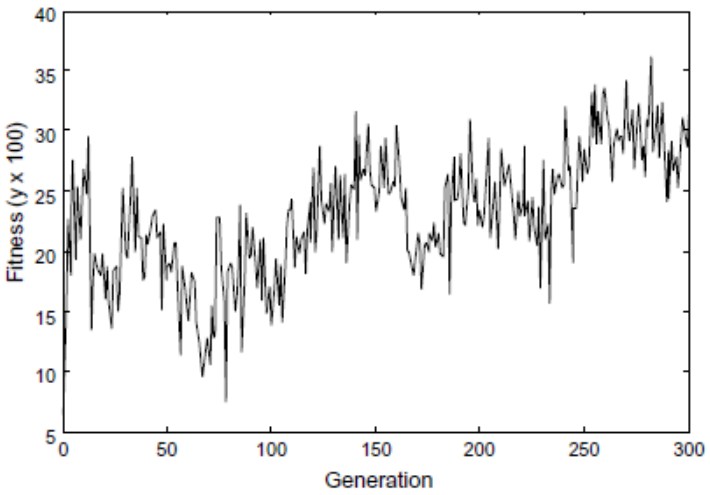


図 5-11 動的環境における進化過程

(2) 相関関係

動的環境における左右のモーター出力の相関関係を図 5-12 に示す．進化の初期段階において静的環境の場合と同様に相関関係はばらばらであった．進化が進むにつれて負の相関関係に収束していったが，全ての個体が収束はしなかった．これは，ロボットが動的環境から多くのノイズの影響を受けていたのと，協調の度合いを小さくとっていたためだと考えられる（$\lambda = 1/18$）．パラメータ λ の影響については後述する．

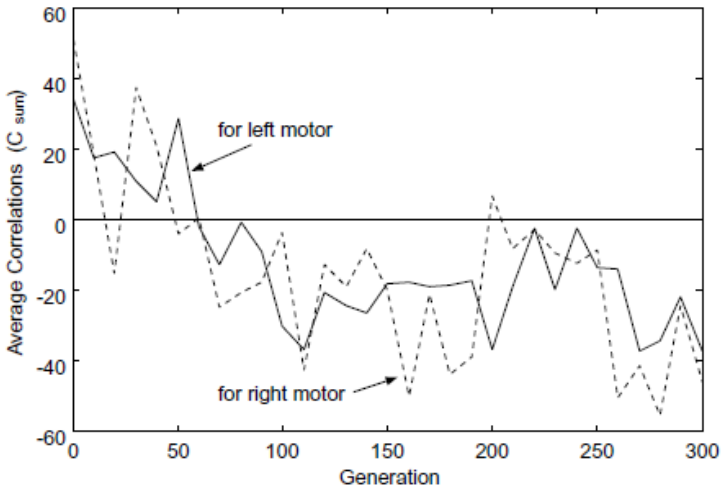


図 5-12 動的環境における相関関係

(3) 結合荷重の機能

図 5-13 に，動的環境における 2 つのコンポーネントネットワークを用いた場合の，10 世代目，50 世代目，100 世代目の結合荷重を示す．2 つのコンポーネントネットワークの結合荷重は，静的環境の場合と同様に機能的違いを示した．初期の世代から 50 世代まではばらばらな値をとっていたが，進化が進むにつれて明確な違いを示し始めた．100 世代目では，結合荷重が正反対になっているのが確認できる．したがって，コンポーネントネットワーク A が障害物を左によける機能を持っているとき，コンポーネントネットワーク B は右によける機能を持っていると考えられる．

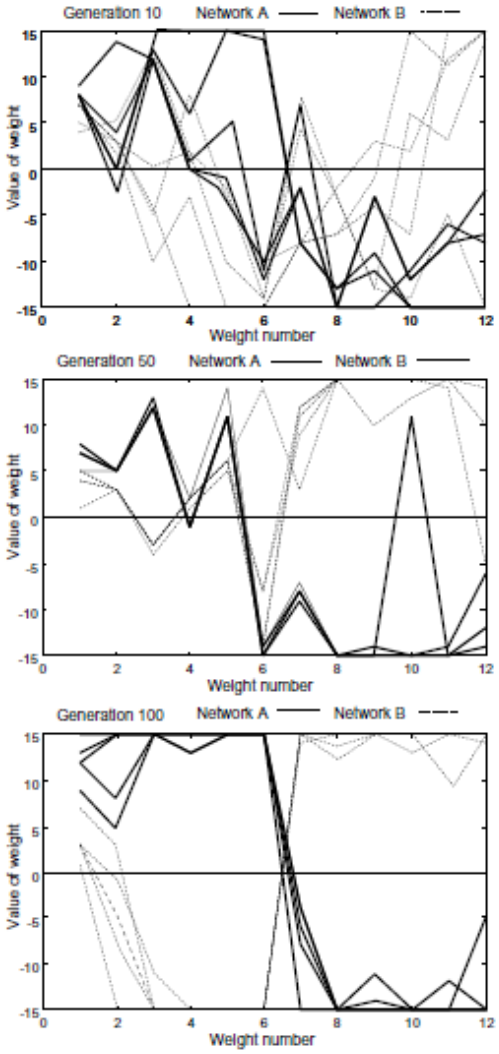


図 5-13　2 つのコンポーネントネットワークの場合の動的環境における結合荷重

5.5. 実験結果の分析

5.5.1. 協調学習の強度の効果

パラメータ λ はコンポーネントネットワーク間の協調の強さの制御を行う．コンポーネントネットワークの間の相関関係は，λ の値を大きくするに従って非協調となる．制御コントローラーの振る舞いはこの値によって影響される．λ の値によって制御コントローラがどのようなふるまいを示すかを確認するため，図 5-14 で示すように 6 つの異なる値で試してみた．

実験結果より，$\lambda = 1/2$ から $\lambda = 1/8$ の間は，評価値の向上が見られなかった．λ がこの範囲にあるとき，2 個のモーターの相関関係はほとんど否定的となっている．これは，実験で用いた評価関数によって制御コントローラーが形成される場合，常に両方のモーターが負の相関関係を維持しているのは良くないということを示している．しかし，λ の値をさらに小さくしていくと，$\lambda = 1/12$ から $\lambda = 1/20$ の間では評価値の向上が見られた．したがって，λ の値は $\lambda = 1/8$ 以下とするのが適当といえる．本研究では，$\lambda = 1/18$ で行った．

図 5-15 は λ の値と 2 個のモーターの相関関係の強さ示している．相関関係の強さは，2 個のモーターすべての相関係数を足し合わせており，-1 から+1 までの値をとる．結果より λ の値が大きくなるにつれて相関係数が強くなっており，制御コントローラとして良くないといえる．また，実験を行っている経過の中で，以下の 2 つの状況に気付いた．

(a) ロボットは，$\lambda = 1/12$ から $\lambda = 1/20$ もしくはそれより小さい値のときに，環境内を障害物回避しながら行動するようになった．これらの負の相関関係は定期的に変化した．左右のモーターの相関関係は正の相関になったり負の相関になったりして，進化が進むにつれて良くない面がなくなっていった．これらには 2 つの理由があったと考えられる．1 つ目は，利用した入出力データは非常にダイナミックであり安定的とは言い難い．2 つ目はセンサーの入力信号は，静的環境においてもかなりのノイズが入力されている．したがって，モーターの出力値は相関関係に定期的に影響を受けていると考えられる．

(b) λ の値が $\lambda > 1/8$ の場合，ロボットの移動が遅くなることが多くあった．また，ロボットが障害物を発見したとき，次の行動を決めるまでに 2,3 秒間停止するような行動を行った．そのため，障害を発見して停止しているため移動を行うことができず，結果として移動距離が少なくなり評価値が減少した．したがって，λ の値は 1/8 未満であるべきである．

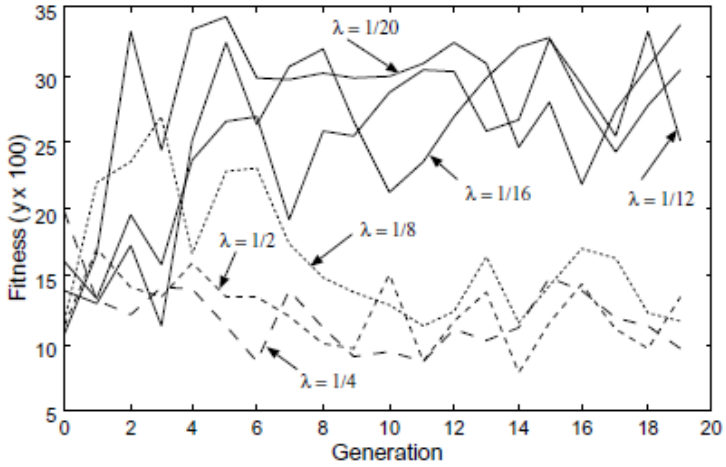


図 5-14　異なったλの値による結果

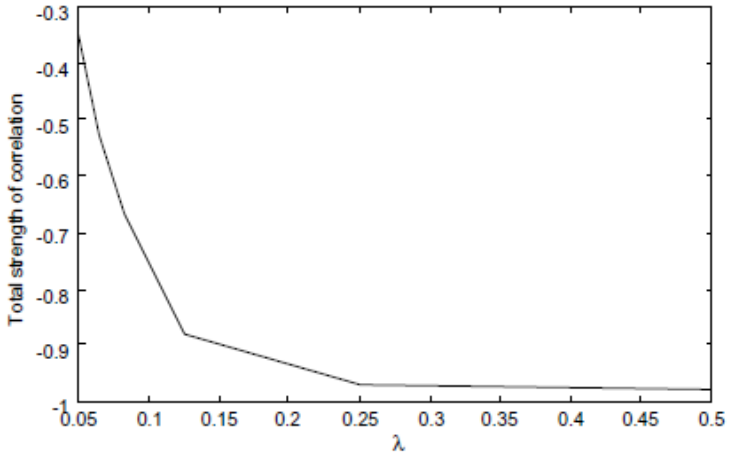


図 5-15　λの値と相関係数の関係

5.5.2. 障害物によるコントローラーの応答

堅牢な制御コントローラーの要件の 1 つとして，外部要因の影響を受けないことがいえる．外部の要因に対して制御コントローラーの有効性を示すために，ロボットが移動中に意図的に人間の介入を行う実験を行った．ロボットが環境を移動中，1 分おきに白い物体をロボットの正面に置いた．この時の進化過程を図 5-16 に示す．ロボットは 21 世代に達する前に障害物回避行動の獲得ができており，外部要因に対して堅牢な制御コントローラー

であるといえる.

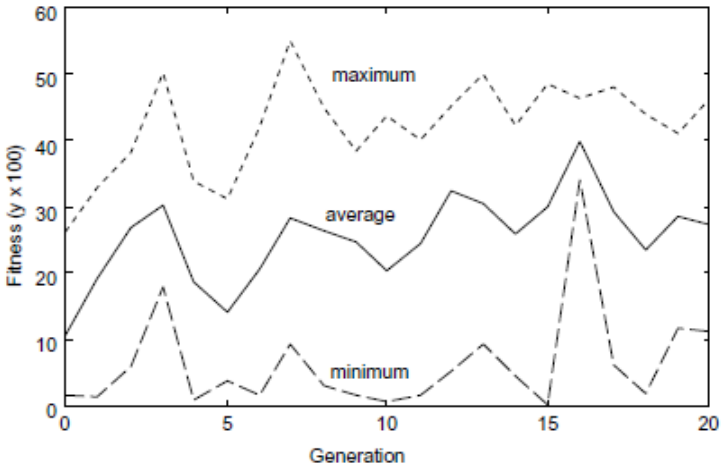


図 5-16　外部要因による制御コントローラーの反応.

5.5.3. 非協調コントローラーでの評価値

図 5-17 に負の相関関係の制御コントローラーと協調しない制御コントローラーの進化過程を示す. 協調を行っている制御コントローラーは, 協調を行わない制御コントローラーより良い結果が出ている. 協調を行っているコントローラーで重要なのは, コンポーネントネットワーク間がタスクを分離して動作している点である. 協調を行わない制御コントローラーの場合は, 何かしらのセンサー情報によりコンポーネントネットワーク同士の結果の相違のために性能が低下する. しかしながら, 性能の低下はすぐに回復している. これは, 協調学習を行わない場合, それぞれのコンポーネントネットワークで認識している環境の詳細が違っているのではないかと考えられる.

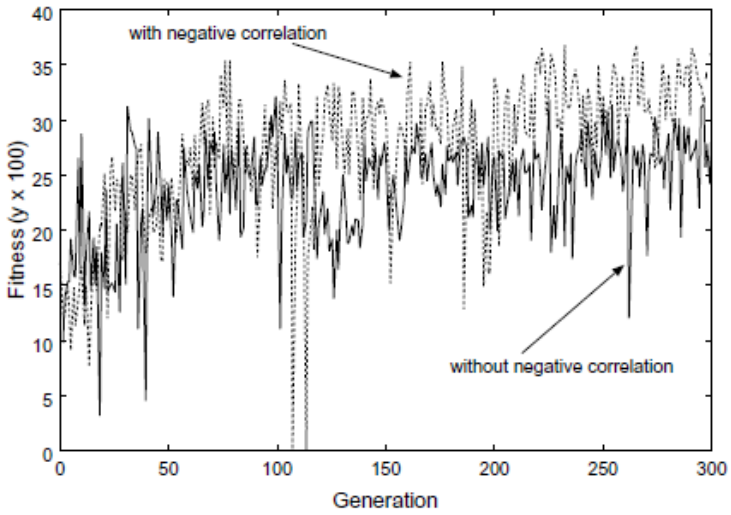


図 5-17　静的環境における非協調学習

5.5.4. 単一ニューラルネットワークと非協調コントローラとの比較

この章で，単一ニューラルネットワークと協調を行わない集合ニューラルネットワーク，そして提案手法である協調を行う集合ニューラルネットワークでの制御コントローラーとしての比較を行う．単一ニューラルネットワークは，ロボットの前方 6 個のセンサーを利用し，1 つのニューラルネットワークを用いている．そのため単一ニューラルネットワークは 12 個の接続を持っている．そのほかのパラメータと遺伝的オペレーションは従来通りである．協調を行わない集合ニューラルネットワークは，簡単なニューラルネットワークの集まりである．

(1)　行動軌道

単一ニューラルネットワークを用いたロボットを一定時間環境内で移動させた行動軌道を図 5-18(A)に示す．ロボットは矢印によって示される軌道 *abcdefijkl* の順で行動した．壁や障害物とぶつかってしまった地点を矢印で示す．特に特徴的な行動はなく，障害物回避行動を行っている．

同じように，協調を行う集合ニューラルネットワークの行動軌道を図 5-18 (B)に示す．ロボットは矢印によって示される軌道 *abcdefijkl* の順で行動した. 壁や障害物とぶつかるこ

となく行動しており，障害物回避行動を行っている．特徴的な行動については，次章にて解説する．

協調を行う集合ニューラルネットワークのための計算時間は単一ニューラルネットワークの場合のおよそ 2 倍である．これは，協調学習のために入出力データを都合 2 回必要とするからである．しかしながら，協調を行う集合ニューラルネットワークは，単一ニューラルネットワークのに比べて速い世代で衝突回避行動を獲得する．協調を行う集合ニューラルネットワークは 20～25 世代以内に障害回避行動を獲得しているのに対して，単一ニューラルネットワークは 40-45 に世代を要している．また，単一ニューラルネットワークは障害物回避行動の獲得後も，まれに壁や障害物に衝突してしまっている．

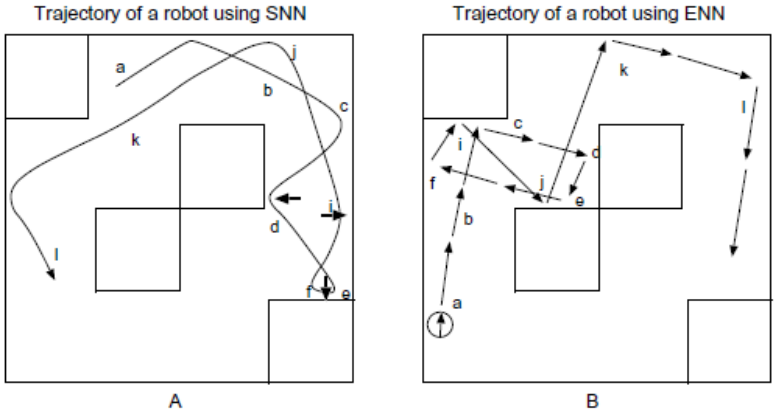


図 5-18　SNN と ENN を用いたロボットの行動軌道

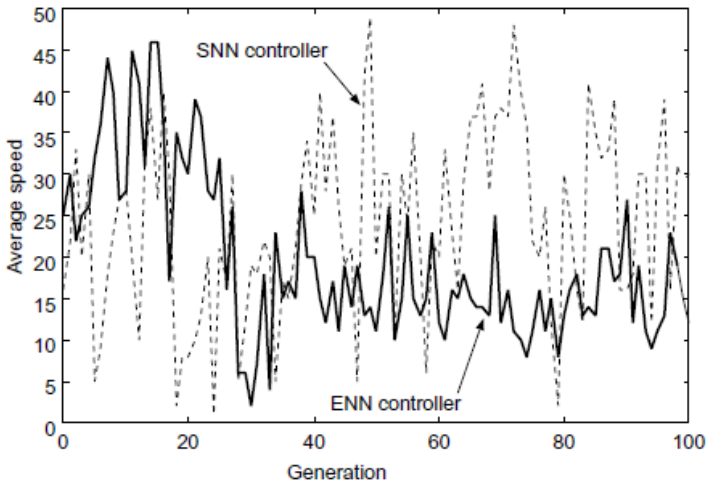


図 5-19　進化過程における平均移動速度

(2)　協調学習による特徴的行動

- 経路反復

  経路反復は，ロボットが図 5-20(A)に示される経路を戻ってくることである．ロボット R は経路 ab に沿って移動をはじめ，W3 の壁まで到着する．そして，今までたどってきた逆の経路 ba で移動を行う．経路反復行動は，協調学習のためにコンポーネントネットワーク間で競合が起こるためではないかと考えている．この競合は，集合ネットワークの訓練中に競合学習によって引き起こされると示されている[27]．各コンポーネントネットワークはなるべく環境の情報を取得したいため，この競合によりロボットを経路反復の行動に導くと考えられる．

- 往復行動からの脱出

  経路反復を始めると，そのうち図 5-20(C)で示すように従来の経路を外れ新しい場所へ行動する．障害物に面した時に，評価関数による個体評価が行われるため，遺伝的オペレーションによる進化の影響を受けたと考えられる．

- 直角に曲がる

  ロボットが直角に障害物を回避するということは，知的な回避行動を示している．壁

や障害物に直面すると少し戻って右か左に回転を行う．この行動を図 5-20(D)に示す．

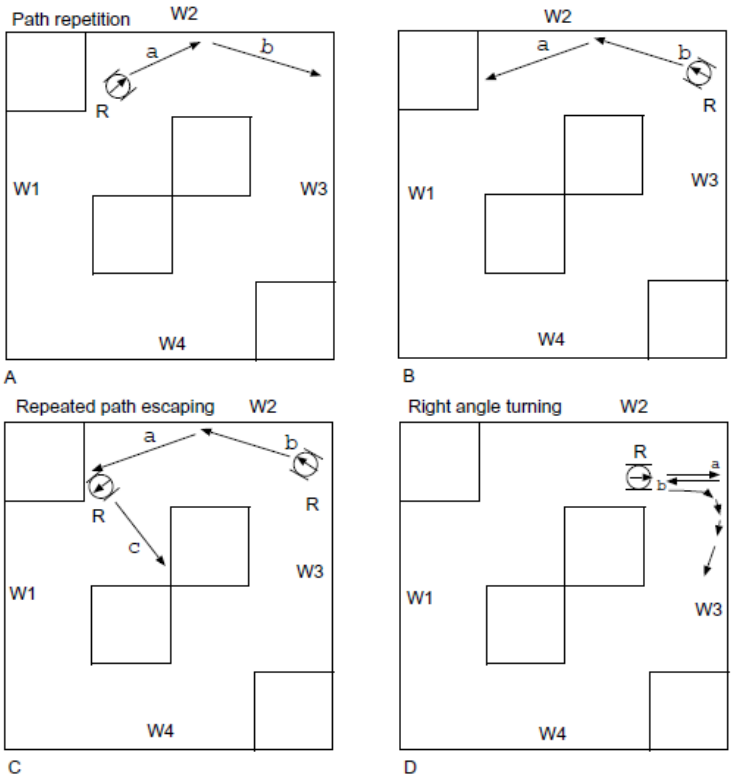


図 5-20　進化ロボットの特徴的行動

(3) 制御コントローラーの堅牢性

異なった制御コントローラーの振る舞いを評価して比較するために，タスクを行う上で重要な性能の品質を示して，制御コントローラーの違いを示すのが重要である．本研究では，タスクの完成速度，衝突率，堅牢性の 3 つの評価軸でロボットの制御コントローラーの比較を行った．

- Sensory motor Loop Value (SLV)

  ロボットが行動を起こすまでにかかる時間は，部分的にホストコンピュータとホストコンピュータからロボットへのデータ転送速度に依存している．構成に評価するため

に，ロボット内のタイムステップを用いた．ロボットの個々のタイムステップは，感覚，処理，行動によって特徴づけられる．このステップでは，センサー入力を変換してモータの出力となる．これらを Sensory motor loop と呼ばれている．本研究では，1 回の Sensory motor Loop に 0.1 秒かかる．Sensory motor Loop の数はロボットがタスクを完了するまでにどれぐらい時間がかかったかを測定するのに利用する．

- Collision Rate (CR)

  壁や障害物との衝突の回数を衝突率として記録した．衝突率が少ないほど，良い制御コントローラーといえる．

- Robustness (R)

  ロボットの制御コントローラーはより少ない衝突率かつ，最小の SLV である場合，堅牢性があるといえる．堅牢性は次式で表す．

$$R = \frac{1000}{SLV \times CR} \qquad (5\text{-}12)$$

これらの 3 つのパラメータは評価関数には利用していない．この 3 つのパラメータは，制御コントローラーの堅牢性を示す評価基準として用いた．R の値が大きいほうが，小さいものより良い制御コントローラーといえる．

単一ニューラルネットワークで一番良い個体と集合ニューラルネットワークで一番良い個体を選び，15 分間テストを行い，5 回繰り返した．CR は 15 分間のうち壁や障害物に衝突した回数となっている．各値を平均したものを表 5-2 に示す．集合ニューラルネットワークのほうが単一ニューラルネットワークにくらべて良い成績となっている．したがって，集合ニューラルネットワークのほうが堅牢性のある制御コントローラーといえる．重要な点として単一ニューラルネットワークに比べて集合ニューラルネットワークは SLV の値が大きくなる点である．これは，集合ニューラルネットワークの速度は 2 つのコンポーネントネットワークの平均値になっているからである．そのため，集合ニューラルネットワークを制御コントローラーとしているロボットは，単一ニューラルネットワークのものに比

べて，少し長い間環境を観察する．しかしながら，集合ニューラルネットワークは単一ニューラルネットワークより壁や障害物に衝突しない．したがって，集合ニューラルネットワークはロボットの制御コントローラーとして単一ニューラルネットワークのものより良いといえる．

表 5-2　SNN と ENN の SLV，CR，R

	SLV	CR	R
SNN	153.57	2.33	2.79
ENN	188.43	0.33	16.08

(4) その他の手法

コントロールシステムとして，協力的なニューラルネットワークを使用するいくつかの方法が提案されている[20,29,30]．Hartono は，いくつかの独立したニューラルネットワークからなるニューラルネットワークの集合モデルの実装を試みている[20]．集合は，例えばセンサー入力をロボットの特定の行動に結びつけるなどの戦略を得るために，自動的に個々に訓練され，予測できない環境に対応するために戦略を切り替えて利用される．一方，本研究の手法では集合の中に機能的に異なった 2 つのネットワークを実装する．したがって，切り替えの仕組みや写像関数などを用意する必要がない．そのような仕組みを，進化の間，協調関係を最小にすることによって自動的に適応する．前途の方法が学習だけを使うのに対し，本研究では学習と進化を利用している．

Goldberg はセンサー入力を共有している 2 つの集合ネットワークで経路制御を行う共同コントロールの実装を試みている[29]．1 つのコンポーネントネットワークの出力をロボットが右へ移動するのに利用し，他方のコンポーネントネットワークの出力をロボットが左へ移動するのに利用している．一方，本研究の手法では，メンバーネットワークのすべての出力を用いてロボットの制御コントローラーで利用している．

5.6. 考察

変化する環境や未知の環境で行動する知的で自律的なロボットに最適なコントローラーを設計するのは，研究者にとって素晴らしい挑戦である．集合ニューラルネットワークは近年研究がはじめられ，コントロールシステムを設計する問題への試みの報告は少ない．本研究では，集合ニューラルネットワークの進化とコンポーネントネットワークの協調の 2 つのプロセスを結合する試みを行った．提案した新しい手法は，実環境で有効なコントロールシステムを構築するための中心的な役割として，集合の協調機能と構築を取り込むものである．

集合ニューラルネットワークに関するレポートはわずかしかないが，集合ニューラルネットワークコントローラーは，我々が知るところにおいて実際の移動ロボットでの協調学習と進化利用する初めての試みである．集合ニューラルネットワークコントローラーは，進化過程における協調学習について調査を行う．2 層もしくは 3 層の単一ニューラルネットワークでは，この種のエキスパート学習の実現は難しい．他の試みとは異なり，集合ニューラルネットワークコントローラーは，コンポーネントネットワーク同士の相互作用を協調学習と進化によって獲得するものである．進化過程と学習課程は，計算量を少なくするために 2 つの段階に分けて行った．ロボットは生存時間内で環境内を行動し，入出力データを収集した後，ロボットの結合荷重の更新を行った．

本研究では，静的環境と動的環境において実験を行った．センサーを通しての入力情報は，静的環境においてもダイナミックであるが，それはロボット自身の行動によるものである．動的環境は，障害物と光量の調整可能な電球を置くことによって実現を行い，コントローラーは外的要因に対しても良い振る舞いを行うことが確認できた．このような協調がより少ない衝突率でコントローラを構築している点で，単一ニューラルネットワークコントローラーや協調しない集合ニューラルネットワークコントローラーに比べて優れているといえる．コントローラーはセンサー情報を前提として知的な動作を行っているため，ロボットは環境を完全に把握し慎重に行動をしているといえる．

進化は適合において非常に遅い手法であることは既知であるので，比較的少ない個体数

を使用した．それらの個体が，効果的に環境を認識し，人間の手助けなしでロボットの衝突回避行動を獲得するということがわかった．また，少ない個体数により計算量も少なく済んでおり，ロボットは与えられた環境においてスムーズな行動を獲得している．進化ロボティックスにおいて重要な問題の 1 つとして，環境内をスムーズに移動することがあげられる．進化は空間における行動を獲得し，協調学習はセンサー情報より現在の事象を捕えている．したがって，ロボットは効果的に環境の構造を知っているということができる．

集合ニューラルネットワークは衝突に対して堅牢であると考えられる．単一ニューラルネットワークは集合ニューラルネットワークに比べて動作が速い半面，壁や障害物に衝突する場合も多い．ロボットの進化後にでも壁や障害物に衝突してしまうと障害物回避行動としては意味を成さない．集合ニューラルネットワークは単一ニューラルネットワークより動作が遅いかもしれないが，コンポーネントネットワークの動作は平均的であり，ロボットは壁や障害物に衝突することなく，慎重かつ知的に環境内を行動する．しかしながら，各コンポーネントネットワークが特定の仕事を行っているかどうかは不明瞭である．これは環境構造の特定個所について調べるのが難しく，センサー値はカオス的で非常に多くのノイズが含まれているためである．

コンポーネントネットワーク間同士で競合が起こっていることが確認できた．両方のコンポーネントネットワークが，特定の仕事を行っていることを把握できているためだと考えられる．これは，単一ニューラルネットワークが，環境の情報を十分に取得できない場合役に立つといえる．また，経路反復行動は機能が異なるコンポーネントネットワークの典型的な兆候ではないかと考えている．最初の行動は特定のコンポーネントネットワークにより決定され，次の行動は別のコンポーネントネットワークにより決定されなければならない．

5.5.1 で述べたとおり，コントローラーの性能はパラメータ λ によって影響された．利用者はこのパラメータを適切な値に設定する必要がある．多くのコンポーネントネットワークを利用すると，代表的な結合荷重と相関関係を可視化しずらいため，よりシンプルな 2 個のコンポーネントネットワークを利用した．

5.7. まとめ

自律行動型ロボットにおける制御コントローラの構造獲得方法として，集合ニューラルネットワークを利用する方法について述べた．協調機能が，相反の関係でコンポーネントネットワーク間同士の協調に利用されたことが確認できた．結合荷重と相関関係の機能は，コントローラーによって示され，左右の出力はほぼ機能的に異なっていた．その結果，個々のネットワークは競争による異なった方法で環境を認識した．

特徴的な行動として，慎重に直角に曲がったり，経路反復を行うなどがロボットの行動中に見られた．これらの行動は予備知識無しで，ロボットの制御コントローラーが環境内でロボットを制御していることが行動軌跡より示せた．また，制御コントローラーの自律的構造化においてユーザによる影響はパラメータのみである．提案手法は，単一ニューラルネットワークと協調を行わない集合ニューラルネットワークとの比較を行ったが，両方の結果より提案手法の結果が良かったことが示せた．進化を伴うより複雑な協調機能の獲得は今後の課題である．

自律システムの構造化において，重要なふるまいの違いを生み出すためにセンサー情報のわずかな違いから鞍点を見出すことが重要である．例えば，ロボットが障害に直面したとき右に避けるか左に避けるかで競合が発生する．負けたほうは，環境に対して少ない適応能力となってしまう．サブサンプションアーキテクチャーはそのような鞍点を作ることができるが，ロボットのデザイナーは環境などの予備知識を持って構造を設計しなければならない．機械学習がベイジアン手法や進化手法によって，鞍点を明確にすることは難しく，むしろ進化や学習の過程で消えてしまう．負の相関関係があるコンポーネントネットワークを用いた提案手法では，別々のコンポーネントに機能を保存することで，鞍点の形成を行うことができたといえる．

実際の移動ロボットを用いた本実験によって，進化の過程においてコントロールシステムとして有効なネットワークを形成することが示せた．これは，進化の過程において順番に鞍点の形成ができる能力を示している．本研究で比較した以外の手法との比較を行った上で，より広い分野での応用のためにさらなる研究が必要である．

5.8. この章の参考文献

[1] D. Floreano and F. Mondada, “Evolution of Homing Navigation in a Real Mobile Robot,” IEEE Transaction on Systems, Man, and Cybernetics -Part B, Cybernetics 26(3), pp. 396-407, 1996.

[2] D. Floreano and F. Mondada, “Evolutionary Neurocontrollers for Autonomous mobile Robots,” Neural Networks, 11, pp. 1461-1478, 1998.

[3] K. Sims, “Evolving 3D Morphology and Behavior by Competition,” Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Ed. R. Maes, pp. 28-39, 1995.

[4] D. Cliff, I. Harvey and P. Husbands, “Exploration in evolutionary robotics," Adaptive Behavior, vol.2, no.1, pp.73-110, 1993.

[5] J. Kodjabachian and J.A. Meyer, “Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects," IEEE Trans. on Neural Networks, vol.9, no.5, pp.796-812, 1998.

[6] T. Fukuda, N. Kubota, and T. Arakawa, “GA Algorithms in Intelligent Robots, in Fuzzy Evolutionary Computation, Kluwer Academic Publishers, pp. 81-105, 1997.

[7] Kim C. Ng and M. T. Mohan, “A neuro-fuzzy controller for mobile robot navigation and multi-robot convoying," IEEE Trans. on Syst., Man and Cybernetics- Part B: Cybernetics, vol.28, no.6, pp. 829-840, 1998.

[8] S. Nolfi and D. Floreano, “Evolutionary Robotics: Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press, Cambridge, MA, 2000.

[9] S. Nolfi, D. Floreano, “Learning to Adapt to Changing Environment in Evolving Neural Networks,” Adaptive Behavior, 1, pp. 99-105

[10] B. Yamauchi and R. D Beer, “Sequential Behavior and Learning in Evolved Dynamical Neural Networks,” Adaptive behavior, 2(3), pp. 219-246.

[11] R. D. Beer and J. C Gallagher, “Evolving Dynamic Neural Networks for Adaptive Behavior,” Adaptive Behavior, vol. 1, pp. 91-122, 1992.

[12] D. K. Pratihar, “Evolutionary robotics - A Review," Sadhana, Vol. 28, part. 6, pp. 999-1009, Dec. 2003.

[13] D. H. Ackely and M. L. Littman, “Interaction between learning and evolution, In Langton, C. G. Taylor, C. Farmer, and S. Rasmussen (Eds.), Artificial Life II, pp. 487-509, 1992.

[14] G. E. Hinton and S.J. Nowlan, “How learning can guide evolution", Complex Systems, vol.1, pp.495-502, 1987.

[15] R. A. Brooks, “A robust layered control system for a mobile robot", IEEE Robotics and Automation, RA-2, pp.14-23, 1986.

[16] Billard and G. Hayes, “DRAMA, a connectionist architecture for control and learning in autonomous robots", Adaptive Behavior, vol.7, no.1, pp.35-63, 1999.

[17] L. K. Hansen and P. Salamon, “Neural network ensembles", IEEE trans. on Pattern Analysis and Machine Intelligence, 12, pp. 993-1001, 1990.

[18] Krogh and J. Veledsby, “Neural network ensemble, cross validation and active learning," In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds), Advances in Neural Information Processing Systems 7, MIT press, 1995.

[19] I. Harvey, “Artificial Evolution: A Continuing SAGA”. In T. Gomi (Ed.), Evolutionary Robotics: Proceedings of International Symposium, ER2001 (pp94-109). Berlin Heidelberg: Springer.

[20] P. Hartono, K. Tabe, K. Suzuki and S. Hashimoto, “Strategy acquirement by survival robots in outdoor environment", Proc. of the IEEE international conference on Robotics and Automation, Sept. 14-19, Taipei, Taiwan, 2003.

[21] S. Nolfi and D. Floreano, “Learning and Evolution,“ Autonomous Robots, 7(1), 1999.

[22] Y. Liu and X. Yao, “Ensemble Learning via Negative Correlation,” Neural Networks, vol. 12, pp. 1399-1404, 1999.

[23] F. Mondada, E. Franzi, and P. Ienne, “Mobile Robot Miniaturization: A Tool for Investigating in Control Algorithm,” Proc. of Third International Conference on Experimental Robotics, Kyoto, Japan, pp. 501-513, 1993.

[24] D. Floreano and F. Mondada, “Automatic creation of an autonomous agent: genetic evolution of a neural network driven robot," In D. Cliff, P. Husbands, J. A. Meyer and Wilson (Eds.), From animals to animates 3: Proceedings of the third international conference on simulation of adaptive behavior (SAB ’94) (pp. 421-430). Cambridge, MA: MIT press.

[25] V. Braitenberg, Vehicles: Experiments in synthetic psychology. Cambridge, MA, MIT press, 1984.

[26] Md. Monirul Islam, and K. Murase, “Chaotic dynamics of a behavior-based miniature mobile robot: effect of environment and control structure", Neural Networks, vol. 18, no. 2, pp. 123-144, 2005.

[27] Md. M. Islam, X. Yao and K. Murase, “A constructive algorithm for training cooperative neural network ensembles", IEEE Trans. on Neural Netw., vol.14, no.4, pp.820-834, 2003.

[28] Md. Shahjahan, “Evolution of Neural Controller for Autonomous Robots in Open Environment", Ph. D Dissertation, University of Fukui, Bunkyo, Japan, 2005.

[29] K. Goldberg and B. Chen, “Collaborative control of robot motion: Robustness to error, IEEE/RSJ Intl. Conf. on Robotics and Systems, October, 2001.

[30] R. C. Arkin, “Cooperation without communication: Multi agent schema-based robot navigation", Journal of Robotic Systems, 9(3), pp. 351-364, 1992.

6. 結論

本研究ではさまざまな課題や問題を解決するシステムに，ニューラルネットワークや遺伝的アルゴリズムといったソフトコンピューティング手法を用いて自律的構造化を行い，現実的な応用として新しい分野での適用を試みた．また，ニューラルネットワークや遺伝的アルゴリズム自体の改善としての新しい試みを実際の問題に適用して有効かどうかの検証も行った．

EC サイトにおける商品レコメンドを行うシステムの構造を，ニューラルネットワークで実現することができた．また，遺伝的アルゴリズムにより学習を行うことで，レコメンドされる商品の多様性を確保しつつ，自律的にユーザの趣向にあったレコメンドを行うことができた．現状では従来手法と同じレベルでのレコメンドが行える段階ではあるが，既存手法に比べてレコメンドを行うためのデータ解析量の大幅な削減ができている．ごく初期のレコメンドにおいては学習が行えていないため，一定の条件を加えたり従来手法との組合せが必要である．また，ユーザ毎に遺伝子情報を準備することで，よりユーザ個別の趣向にあったレコメンドを行うことが可能である．レコメンドエンジンは，EC サイトでは必須の機能となってきておりさらなる機能改善が望まれている．本研究で提案を行ったソフトコンピューティング手法により，より個人の趣向にあったレコメンド，データ解析量の削減によるリアルタイムなレコメンドなど，従来手法に加えて新たな手法として期待できる．

Java 言語の難読化において，ニューラルネットワークによってメソッド名のマッピングを行うシステムの構造の獲得ができた．ニューラルネットワークにより実行時に動的にマッピングを行っているため，逆アセンブルをしたソースコードからはどのメソッドが呼び出されているか類推することは困難である．本研究の手法により，実行速度を犠牲にすることなく従来手法より逆アセンブルが難しい難読化が実現できたといえる．実際に利用するには，事前にメソッド名の符号化が必要なため，符号化ツールの準備が必要である．また，従来手法と組み合わせることでさらに強固な難読化が期待できる．近年では Web アプリケーションによるソフトウエア提供が徐々に始まってきているが，作成したソフトウエアの配布はまだまだ必要である．本研究で行った Java 言語の難読化により盗用やセキュリティ問題などを危惧せずに，有用なソフトウエアの流通に貢献できれば幸いである．

進化型ハードウエアを自律行動型ロボットの制御コントローラとして適用し，再構成可能な論理素子が制御コントローラとして構造化されたことを確認した．また，遺伝的アルゴリズムによる進化で障害物回避行動を行う構造の自律的獲得ができた．実際には自律行動型ロボットが行わなければならない問題は絶えず変化しており，ロボット全体の制御はソフトウエア的に行う必要がある．しかしながら，ロボットの障害物回避行動や二足歩行ロボットにおける姿勢制御など，いわゆる反射的に行う動作は俊敏性が要求されるためハードウエアで制御したほうが有利であるのは明白である．ロボットが行動を行う実環境は変化を伴うことから，制御構造も自律的に環境に適用していく必要があるため，進化型ハードウエアによる制御が有用ではないかと考えられる．

協調学習を行う集合ニューラルネットワークによる，衝突物回避行動の行う制御コントローラの構造の獲得ができた．また，個々のニューラルネットワークが別々の機能を獲得し，自律的に協調を行いロボット全体を制御する制御コントローラの構造化が確認できた．本研究では衝突物回避行動の獲得を行ったが，より複雑な協調動作による行動の獲得は今後の課題である．1つ1つのニューラルネットワークが機能や記憶をつかさどることにより，大規模な集合ニューラルネットワークによる複雑な問題に対応する自律行動型ロボットの制御コントローラとしての利用が期待される．

以上の結果より，さまざまな問題を解決するシステムにおいて，ソフトコンピューティングを用いた手法により，問題解決のためのモデルを自律的構造化し，新たな解決手法として有効であることが確認できた．ソフトコンピューティングの現実的な利用は，まだ限られた分野での利用にとどまっているが，本研究の試みにより新しい分野でのソフトコンピューティングによる改善，解決の可能性を示せたのではないかと思う．

謝辞

本研究の追行および論文をまとめるにあたり，終始ご指導ご鞭撻を賜りました福井大学工学部知能システム工学科の村瀬一之教授に深く感謝申し上げます．また，数々のアドバイスをいただいた福井大学工学部知能システム工学科生体システム研究室の諸先輩方はじめすべての皆様にお礼申し上げます.

本論文の作成にあたり，ご協力をいただいた Md. Shahjahan さん，園山あゆみさん，また，格別のご配慮をいただいた株式会社スタートトゥデイの前澤社長はじめすべてのスタッフの皆様に心から感謝いたします.